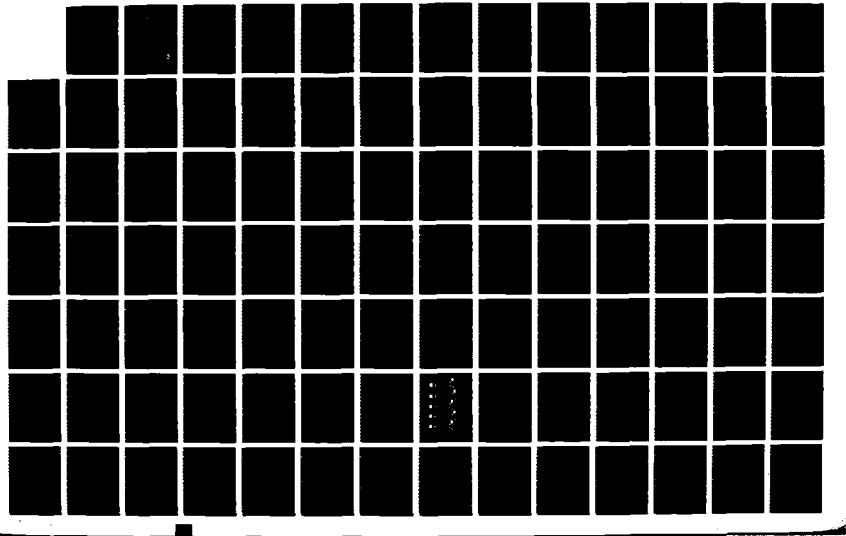
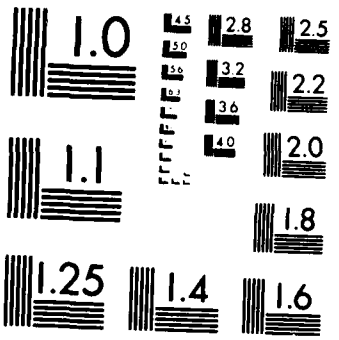
AD-A188 833 MODIFICATION IMPLEMENTATION AND EVALUATION OF A REMOTE TERMINAL EMULATOR A..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C J RIESBERG 1/2

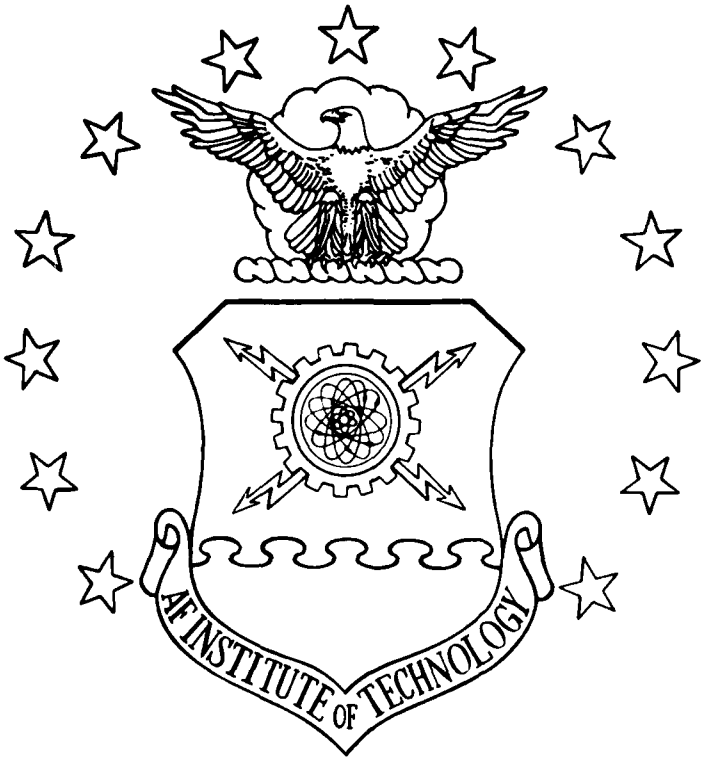
UNCLASSIFIED DEC 87 AFIT/GCS/MA/87D-5 F/G 12/5 NL

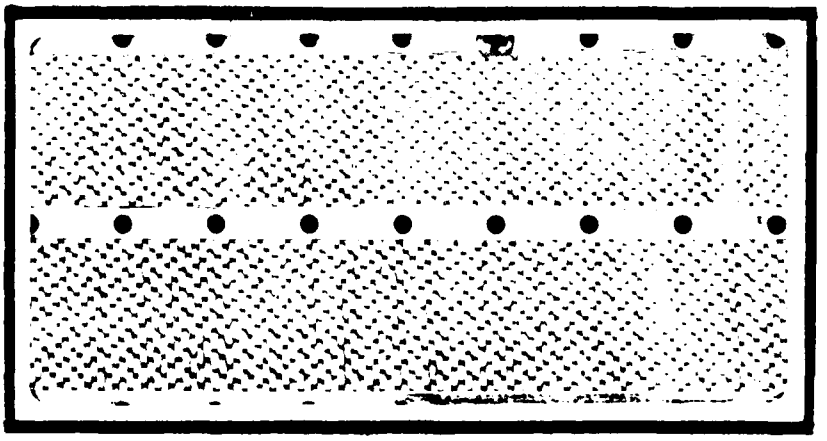




MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

AD-A188 833





DTIC
ELECTE
FEB 1 0 1988
S D
C E

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved for public release and sale; its distribution is unlimited.

88 2 4 045

AFIT/GCS/MA/87D-5

MODIFICATION, IMPLEMENTATION, AND EVALUATION OF A REMOTE TERMINAL EMULATOR AS A SOFTWARE VALIDATION AND STRESS TESTING TOOL

THESIS

Craig J. Riesberg, B.S.
Captain, USAF

AFIT/GCS/MA/87D-5

DTIC
ELECTE

Approved for public release; distribution unlimited

AFIT/GCS/MA/87D-5

MODIFICATION, IMPLEMENTATION, AND EVALUATION OF A REMOTE TERMINAL EMULATOR AS A SOFTWARE VALIDATION AND STRESS TESTING TOOL

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Craig J. Riesberg, B.S.

Captain, USAF

December 1987

Approved for public release; distribution unlimited

Preface

The purpose of this study was to provide the sponsor with a new tool to be used in improving the reliability of their software. In the past, the sponsor was limited to using an internal test driver to exercise their software. This effort allowed the system test director to expand beyond the world of single-threaded validation testing, opening a new window for them in the area of software validation.

In preparing my thesis, I have received considerable help from others. I am very grateful to my thesis advisor Dr. Panna B. Nagarsenker, for her support and guidance during the entire effort. I would also like to thank my committee members, Dr. Brahmanand N. Nagarsenker and Major Duard S. Woffinden for their assistance. A word of thanks also goes to the sponsor's personnel, who aided me in the development and evaluation of the test scenario. Finally, I would like to thank my family, my wife Gail, for her loving support during graduate school, and my daughter Dawn, who understood why her dad could not attend all of her athletic events.

Craig J. Riesberg

Table of Contents

		Page
Preface		ii
List of Figures		v
List of Tables		vi
Abstract		vii
I.	Introduction	1
	Background	1
	Problem Statement	8
	Scope	10
	Assumptions	11
	Approach	12
	Materials and Equipment	14
	Other Support	14
II.	Literature Review	16
	Need for Testing	16
	Acceptance Testing	19
	Validation Testing	21
	Summary	23
III.	Modification and Implementation	24
	History	24
	Modifications Due to Systems Software	25
	Modifications to RTE Software	27
	Hardware Implementation Details	30
	Testing	35
	Summary	36
IV.	Evaluation as a Software Validation Tool	37
	Introduction	37
	Stimulus and Response Collection	38
	Script Assembly	41
	Emulation	44
	Summary	47
V.	Evaluation as a Stress Testing Tool	48
	Introduction	48
	Concepts of Stress Testing	48
	Collection for Stress Testing	50
	Stress Test Emulation	54
	Problems Encountered	61
	Summary	63

VI. Conclusions and Future Studies 64

Conclusions 64
Future Studies 65

Appendix A: Sponsor's Release Data 1984 - 1985 . . . . A - 1

Appendix B: Examples of Inputs and Outputs . . . . . B - 1

Appendix C: Printouts of RTE Generated Files . . . . . C - 1

Appendix D: Screenfaces for Response Verification . . D - 1

Appendix E: Reports Generated During Stress Test . . . E - 1

Bibliography . . . . . BIB - 1

Vita . . . . . VITA - 1

List of Figures

Figure		Page
1.	CAPS Operational Sites During Prototyping	2
2.	Current CAPS Operational Sites	3
3.	Validation Testing Diagram	9
4.	Patch Cable Schematic	31
5.	Revised Crossover Cable Schematic	32
6.	CLM User File Entries for Two Terminals	34
7.	Diagram of Remote Terminal Emulator's Data Flow . .	39
8.	Diagram of Data Flow for Two Terminals	52
9.	Diagram of Sponsor's Computer Performance Evaluation Subystem	57
10.	Current System Configuration	67
11.	Potential Future Systems Configuration	69

List of Tables

Table		Page
1.	Results from Stress Test Emulation	56

Abstract

This project involved the modification, implementation and evaluation of a remote terminal emulator (RTE) as a software validation and stress testing tool. The work, sponsored by the Directorate of Integration and Standardization, Military Airlift Command, utilized a Honeywell DPS-6 Model 95 minicomputer as the host for the RTE application package.

Considerable background information is provided about the sponsor's release environment to show the sponsor's need for a reliable testing tool. Information is also presented about acceptance and validation testing.

The emphasis of this research centered on a RTE made available by the Joint Data System Support Center. This software package, capable of emulating multiple users, needed major modifications before implementation on the sponsor's equipment. After successful implementation, the study examined two areas of use for the sponsor.

The modified package was found to be a very good tool for software validation. Comparisons are made between the RTE and the sponsor's internal test driver in the areas of scenario collection and management. The two tools are also compared during the emulation phase of software validation.

The RTE package was also examined as a stress testing tool. Several problems in the RTE application software which inhibit its use as a stress testing tool are discussed.

MODIFICATION, IMPLEMENTATION AND EVALUATION OF A REMOTE TERMINAL EMULATOR AS A SOFTWARE VALIDATION AND STRESS TESTING TOOL

I. Introduction

Background

The implementation of this remote terminal emulator is the continuation of an effort in software and hardware configuration management which began at the Military Airlift Command in 1982. At that time, the Configuration Management Board in charge of hardware and software for the 1500 Computer Services Squadron, tasked the Directorate of Integration and Standardization with preparing a way to adequately control the organization's computer resources. The directorate designed the MAC Automated Release Facility (MARF) to serve as an overall automated tool to manage the hardware and software of a distributed processing network. MARF was built to control equipment and code for the Consolidated Aerial Ports Subsystem (CAPS) which in 1982 was being prototyped at Travis AFB, Dover AFB, McGuire AFB, and Rhein Mein AB, Germany. With the system development units located at Scott AFB, Ill, the entire distributed network appeared as shown in Figure 1. The system currently runs at eighteen operational sites worldwide as shown in Figure 2.

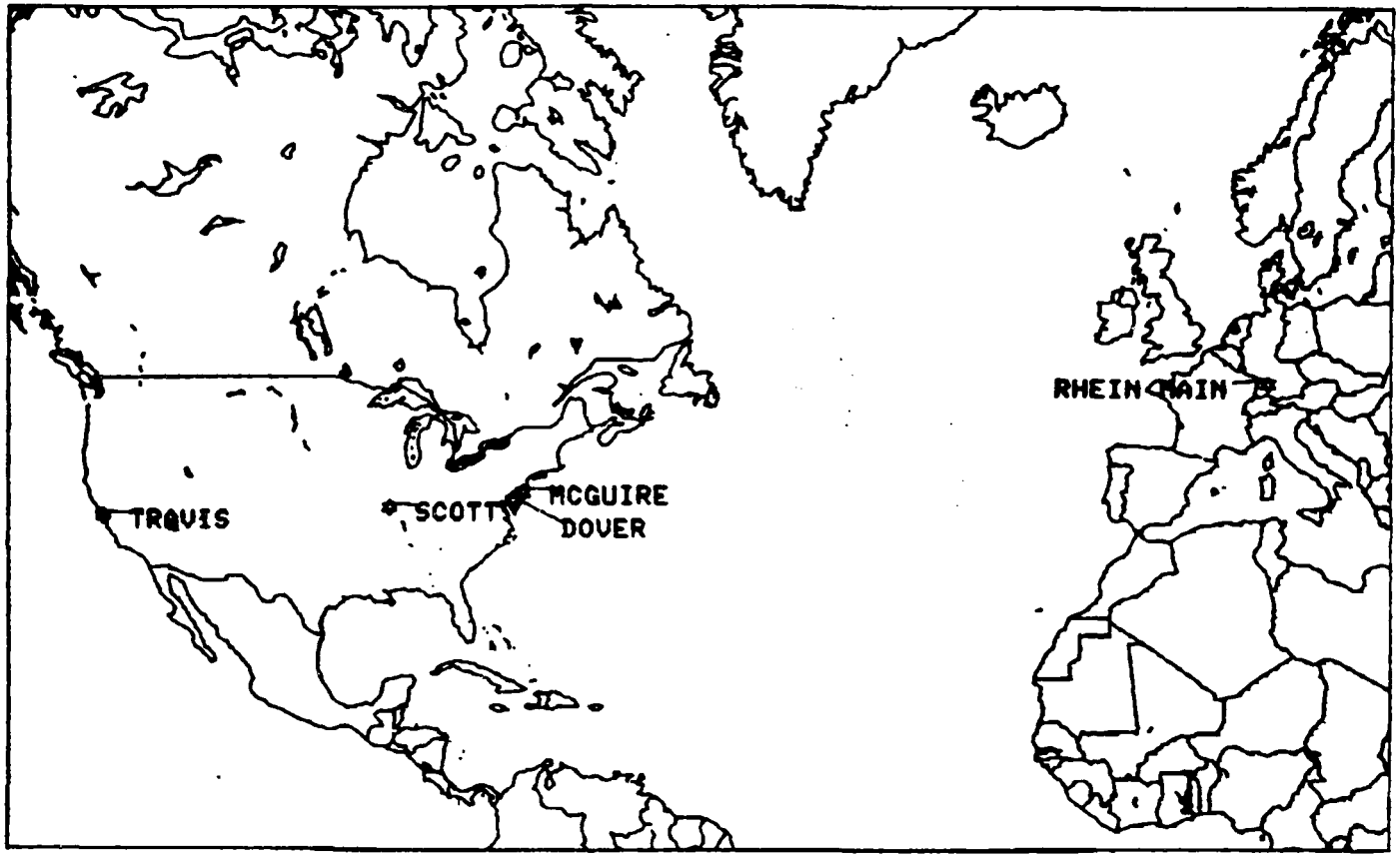


Figure 1. CAPS Operational Sites During Prototyping

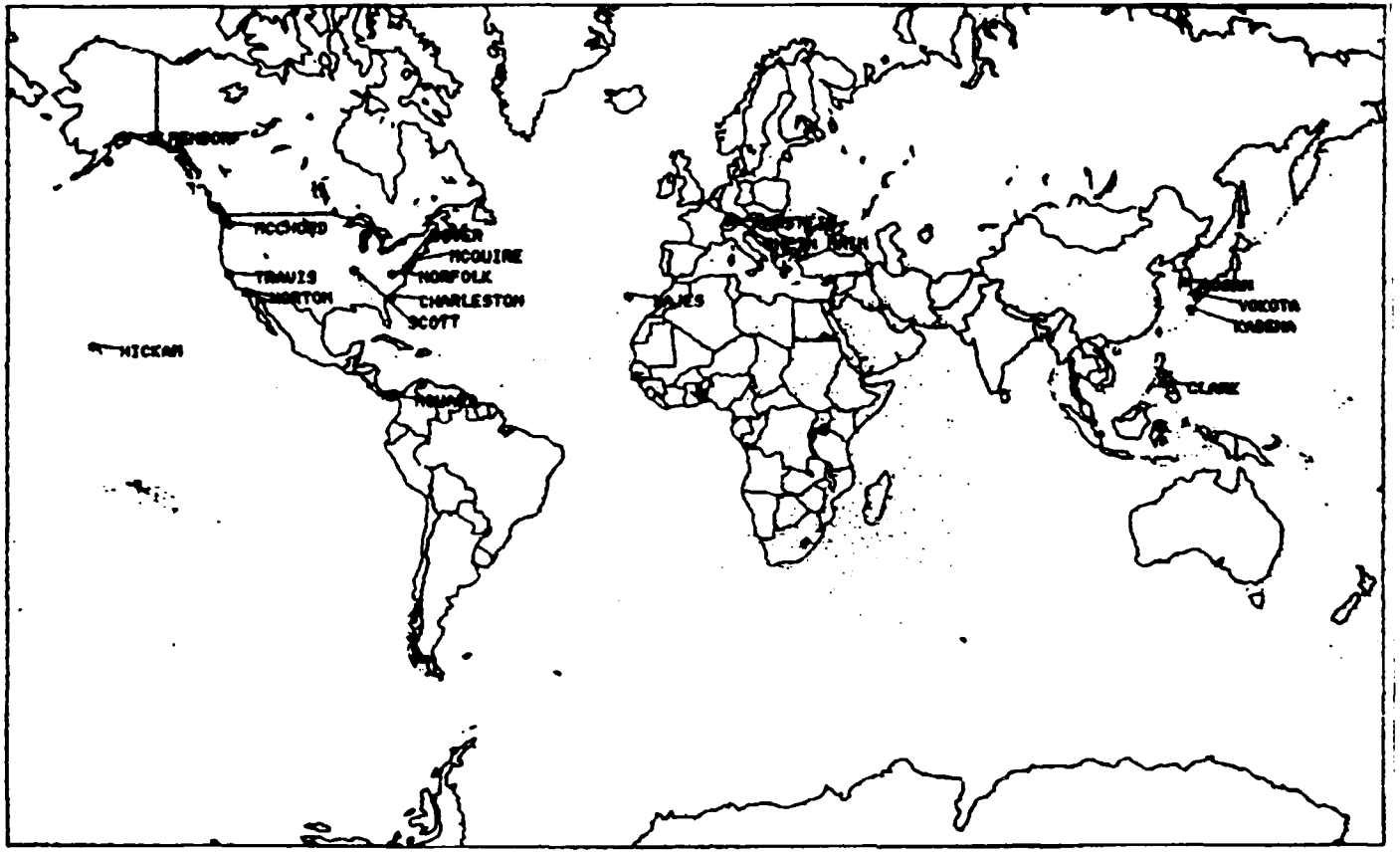


Figure 2. Current CAPS Operational Sites

At least three other factors significantly contributed to the software configuration management problem of this network: 1. the number of releases, 2. the size of the modules and 3. the location of the processing nodes.

The first factor is the sheer number of releases which are being sent to the individual sites. In an effort to keep the software identical at all operational sites, each release must be distributed throughout the network, regardless of which site requested the change. The system went straight from a prototype environment to full implementation. This meant that the system was still in a state of change when it went operational in 1984. A 'try it and see' mentality had been encouraged among the software developers during the prototype phase. As the system went operational and the number of sites increased, the software developers were being asked to revert to more traditional test and distribution techniques.

Data taken from the release control log books of the sponsor shows 68 software releases in 1984, 50 in 1985, 45 in 1986 and 34 in the first half of 1987. The actual release information, divided by system, can be found in Appendix A. The decrease in the number of releases has occurred both because of the maturity of the system as well as the realization by the configuration manager that the system's software could not be adequately tested as fast as it was being released.

The second contributing factor is the size of the software modules being sent to the sites. Without even considering the Honeywell operating system, which is updated quarterly, the user has written 196 different executable modules. The modules are built from 2,765 different source code files. Fifteen of these modules are very large. The module size can best be described by example. The modules are composed of a root which can be up to 32K words, and an addressable overlay area of 32K words. Many of the functional code modules would not fit into a single overlay area and had to be divided into two or more overlays. The largest of the modules has over 200 of these overlays. The module must be delivered in a complete package with all overlays included.

The third and final factor is the limited means of distribution available to the configuration manager. Physical distribution of a release tape to the European, Pacific and Asian theater of operations was found to take up to three weeks. In addition to the unacceptable delay, tapes never arrived at all sites. As long as a single site did not receive the tape, all sites had to wait until the releases was redistributed to the single site. For physical distribution, no commercial overnight or second day delivery system could be found to most overseas sites. Military aircrew members traveling to the sites are not used to distribute tapes unless the tapes contain classified

information. The release tapes do not contain classified information, so that type of distribution is not available to the sponsor.

An electronic transfer system is available in the distributed net, but it has two severe limitations, neither of which can be reasonably eliminated. The first limitation is that all distributions go through a central host. A message sent to this host can only be directed to six different nodes. To send to 18 sites requires the message to be sent to the host three times. The host also has a limitation of six files per message. Therefore, to send 24 of the 196 modules to all 18 sites would require 12 (4 times 3) transmissions to the host.

The second severe limitation of the electronic transfer is the speed of the transfer. The message is broken into packages called unit-blocks and the system is capable of transferring one unit-block per second. When coupled with the size of the bound units discussed as the second factor, the limitation is created. The largest bound unit is broken into 1600 unit-blocks. To transfer it requires over 26 minutes (1600 / 60). Messages to the nodes are not interleaved, so a single release transmission can tie up the communication channel of an operational site for hours.

All of these distribution problems quickly pointed to the need for a good configuration management system. The emphasis was soon to be placed on insuring the quality of the

released software. It was hard enough to release the software once; if it had to be corrected after release, the problem was doubled.

Hardware for both the operational sites and the automated release facility are the same. Current operational sites have from one to three systems consisting of Honeywell DPS-6 model 95's with at least two tape drives, three disk drives with removable packs, two megabytes of memory, and a commercial instruction code processor. All system security measures have been removed from the operating system including passwords for files and logon requirements.

The sole purpose of the operational systems is to provide either passenger reservations service (PACS), cargo management service (CARGO) or enhanced airlift report service (EARLO), or any combination of these services. The need for more than one set of hardware is dictated by the capacity of the machine that is currently providing the service. The operational sites have no compilers, linkers, or any other means to modify or create software. These sites have no computer programmers or analysts assigned to manage or control software. All software comes from the central release facility at Scott AFB, through the automated release facility.

Captain Enrique G. DeJesus introduced the MARF system as a configuration management tool at the Computer Related Information Systems Symposium, sponsored by The Department of

Computer Sciences, US Air Force Academy in February, 1983.(3:3.1-3.22) A refined model of the system was again presented at the Computer Performance Evaluation User's Group 19th meeting in San Francisco, California in 1983.(4:187-196)

Final first cut modifications to the system were completed in 1983 and the final product was presented at the 14th international conference of the Computer Measurement Group, 8 December, 1983, in Crystal City, Virginia.(5:222-230)

The system addresses four elements of configuration management: change control, validation testing, inventory management, and software distribution. The remote terminal emulator will be an important addition to the validation testing module. Figure 3 shows the position of the remote terminal emulator in reference to other elements of the validation testing module.(3:3.13)

Problem Statement

The validation testing leg of the MARF system uses an internal test driver to pass precollected screen faces to application bound units whenever a system is tested. This internal test driver is only able to pass one transaction at a time to the system under test (SUT). After passing an input screen, it must wait for the output from the SUT to verify that the output is correct, before proceeding to the next transaction. Therefore, the internal test driver will not allow tests of two individual bound units (application programs) running simultaneously. While path testing in an

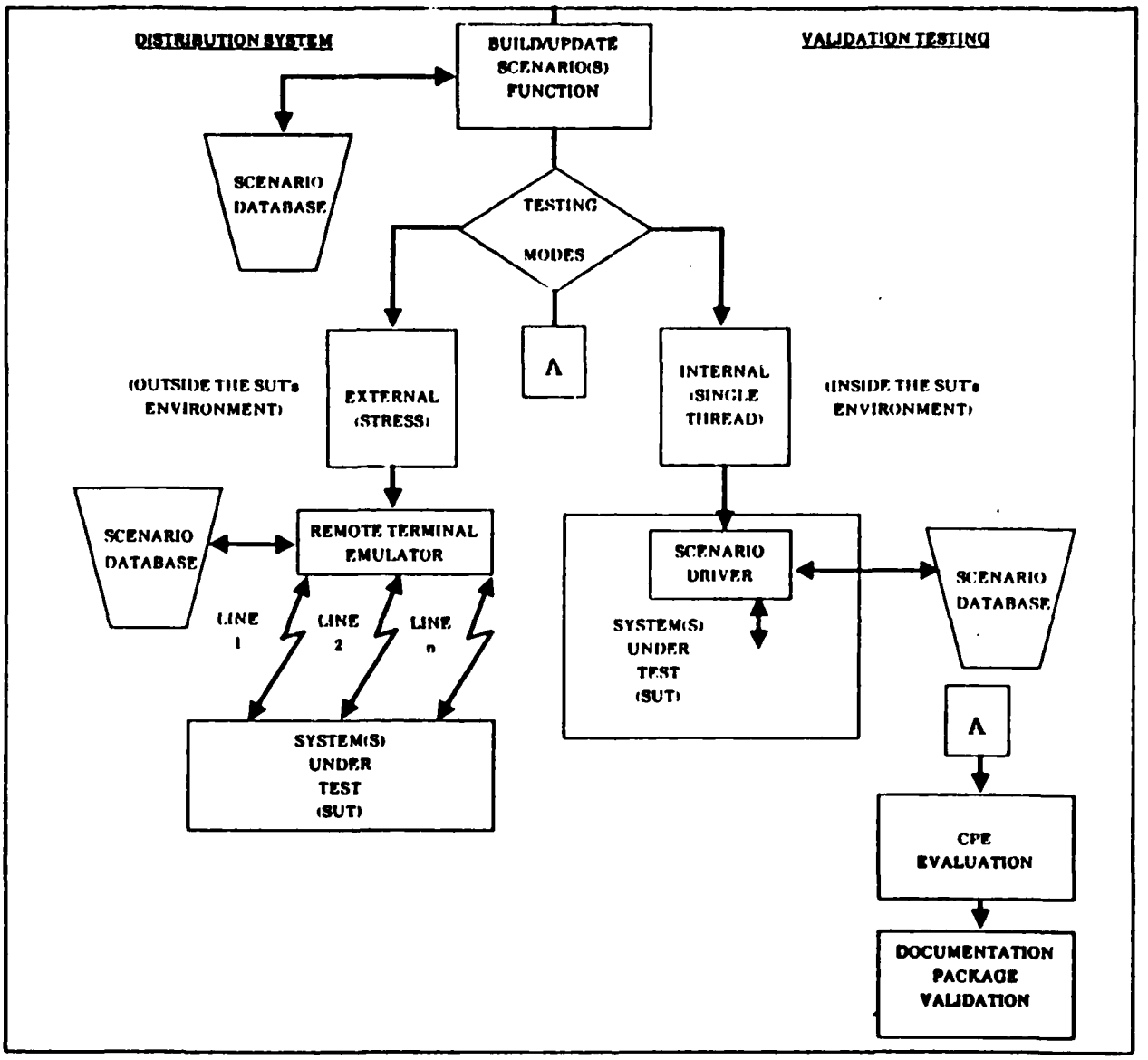


Figure 3. Validation Testing

isolated mode is better than no testing at all, the internal test driver still leaves a large margin for error. The remote terminal emulator allows the system test director to feed inputs to more than one application unit at the same time.

The main problem in integrating the RTE into the MARF system concerns the length of the inputs and outputs. The only RTE which operates on the system hardware is limited to inputs and outputs of 72 characters each. The system needs a RTE which will operate with full screen faces or 1968 characters (24 lines x (80 characters + 2 control characters)). The screen size is the same for both input and output. In addition to the need to modify for input and output sizes, the RTE and the driving system are currently built for different versions of the operating system.

The problem for this study is to implement the RTE and evaluate its use as a validation tool for these systems. If the RTE is an acceptable tool, how does it compare to the internal test driver for use during the different phases of software development, and are there other uses for the RTE?

Scope

This effort consisted of implementing the RTE in the full screen face mode and evaluating its usefulness after implementation. This required updating the RTE package into a system which would operate under the currently running operating system. The connections between the two systems

(RTE implemented system and the system under test) were operated with at least four terminals being simulated. This required either four different lines or some type of alternate polling on a physical line.

The initial implementation was not expected to do data collection in full screen mode, relying instead on the information from the internal test driver as the starting test data base. In addition, the collection scenario which was used in testing the RTE was limited to entries which do not change the corporate data base on the system under test. This was to insure that the collection scenario could be run repeatedly without restoring the data base of the SUT. If the stimulus transactions change the data base, then the data base would have needed to be restored prior to a repetition of the stimulus.

This effort was intended to show and evaluate the use of the RTE as a validation and testing tool, not to do actual software validation for the sponsor. Every effort was made to leave the sponsor with a tool that they could use to test and evaluate their systems. To achieve this goal, some elementary training of the sponsor's personnel was required after implementation.

Assumptions

It was assumed that the people using the RTE package were more than familiar with the concepts of software validation. More emphasis was placed on the correct operation of the RTE

package than on the ease of interaction with the system user. User's manual documentation changes were implemented by pen and ink changes tc the current contractor supplied manual. It was also assumed that the RTE users would be familiar with the application software being used on the SUT.

Approach

The implementation and evaluation of the RTE package took place in four phases. The first phase was the physical connecting of the two machines, coupled with the implementation of the 72 character RTE with the latest operating system. The second and third phase was the evaluation of the package for its intended use of validation and stress testing. The final phase was training the sponsor's personnel.

The first phase was evaluating what lines on the two hardware systems needed to be connected to enable the two DPS-6's to communicate with each other. Extensive changes were needed in the Configuration Line Manager (CLM) User's file. This is the file that determines the input and output channel speeds and the establishment of which unit will be the host and which will act as the slave. Considerable work was necessary to cross the lines between the machines so that the output from A becomes the input to B. Data scopes were necessary to monitor the lines.

After the lines were connected, the phase continued with implementing the RTE package using the 72 character software

and the latest versions of the operating system. This phase of the implementation was the largest unknown. The conversion over several versions of the operating system could have been as easy as recompiling all of the source code and relinking using the latest operating system version. This was not the case. The sponsor's experience with COBOL over the same span of operating systems showed that extensive work was going to be required. To convert the three systems running at the operational sites (PACS, CARGO and EARLO) to the current version, from the version that the RTE is in, took a team of analysts from each system close to six months of work. The RTE is written mostly in FORTRAN and the sponsor had no FORTRAN experience. In addition, the latest version of the operating system uses a different structure of memory pools which generated problems for the RTE package.

The second phase was the evaluation of the package as a full screen software validation tool. During this phase the primary focus was on the software's capability to capture and retransmit inputs and outputs of up to 1,968 characters. It was also during this phase that the software was evaluated to determine if the response from the SUT was identical to a previously accepted response.

The third phase determined if the RTE package was an acceptable stress testing tool. For this work, stress testing is defined as the ability to determine the correct execution of software when it is placed under heavy load. To

fulfill this requirement, the tool needed to collect inputs and outputs from more than terminal simultaneously and be able to retransmit stimuli at least fast enough and to enough terminals to load the SUT.

The final phase of this effort, was to insure that the sponsor was left with trained personnel, qualified to continue the research without 'reinventing the wheel'. To achieve this goal, all of the preliminary work needed to implement and test the system was not only briefed, but thoroughly explained to the sponsor's workers.

Materials and Equipment

The work required two Honeywell DPS-6 Model 95 computers, side by side, with a minimum of 4 input-output ports that could be connected. The equipment is currently in place at Scott AFB and was made available by the sponsor for this work. TDY funds for the work were also supplied by the sponsor. A one week TDY, early in the work, to the RTE developers in Reston, Virginia was used to evaluate the original RTE package to determine what modifications would be needed. Data scopes were available for the equipment and the lines to cross input to output were manufactured during the TDY at Reston. These lines were taken to Scott AFB, Ill and were left with the sponsor for future use.

Other Support

Limited programming support was supplied by the sponsor, especially for the output verification phase. The sponsor

also requested support and cooperation from the contractor currently responsible for maintenance of the RTE. Because of the lack of a Fortran compiler on the sponsor's machine, all of the RTE modifications had to be coordinated and made by the contractor's personnel on their equipment.

II. Literature Review

Need for Testing

There are four major elements of configuration management: change control, validation testing, inventory management, and software distribution. This discussion is limited to the areas which cover software validation testing. Software validation concerns itself with the final usefulness of the software product.

Software validation is the most critical leg in the configuration management scheme. With the current impetus toward distributed processing, changing bundled software at remote processing sites becomes an ever increasing problem. Not only is it necessary to transfer new software to the geographically separated locations, it is important that the software performs as expected. Software validation is the final chance for the releasing authority to insure that the product fulfills its intended use.

Definitions of what testing actually is, are almost as varied as the number of books or articles read. One overall explanation of current software testing was presented by Plessey MicroSystems in a product introduction booklet. They stated:

> The testing of software in a large system is a inexact science at best. Current testing methods are laborious, imprecise, and expensive. These methods usually include single-threaded functional testing and perhaps a more formal verification and validation effort.(15:1.1)

Not knowing or understanding exactly what testing is would not be an insurmountable hindrance if we could be sure of the final result of that testing. They continued in their explanation:

> Overall system testing is usually done in a benign environment with a relatively light load. The connection between a system's test results and its performance in its real environment is tenuous. As it is usually accomplished, the performance testing of software is analogous to testing an aircraft by reviewing its blueprints. Would the reader want to fly in such a machine? (15:1.1)

While everyone agrees that testing is a necessary evil, there are different views as to how important it is and consequently how much effort and system resources should be expended towards adequate testing. In his book on software engineering and design, Zelkowitz says:

> Verification and validation (module and integration testing) of a system occupies about half of the development time of a project. Many debugging aids were developed to lessen this effort; most are implemented as programs to test some feature of a system.(17:27)

What is truly needed for an organization is a tool which is useful not only during the development of a particular software system, but one that is useful over the entire life of the system hardware and beyond.

Just as the system controller can decide how many resources will be devoted to thorough testing, he also must decide within those resource limits, what type of effort he is willing to accept. Often what is considered to be a correct execution depends on the stage of the software in the

life cycle. Zelkowitz referenced Conway in his book, and the subject was called correctness instead of testing.

> ...the term "correct" can have many interpretations. Conway lists eight different meanings for a correct program:
>
> 1. A program contains no syntactic errors.
>
> 2. A program contains no compilation errors or failures during program execution.
>
> 3. There exists test data for which the program gives correct answers.
>
> 4. For typical sets of test data, the program gives correct answers.
>
> 5. For difficult sets of test data, the program gives correct answers.
>
> 6. For all possible sets of data which are valid with respect to the problem specification, the program gives correct answers.
>
> 7. For all possible sets of valid test data and all likely conditions of erroneous input, the program gives correct answers.
>
> 8. For all possible input, the program gives correct answers.(17:27-28)

The level of testing, then depends not only on the resources that are available to test the software, but on the need for correct execution.

Validation of software can be divided into two areas. The validation that occurs on a new system, after design and coding, but prior to or in conjunction with its first implementation, is usually referred to as acceptance testing. While this area is obviously important, this review also covers the continued validation of operational systems. This validation occurs after a change to the baseline software.

This chapter presents different views of software validation represented in current literature. Examined first are those views applying to acceptance testing and then those that deal with continued software validation of operational software.

Acceptance Testing

> The crucial stage in any procurement is the moment when the vendor offers the product to the buyer for inspection to determine whether or not the contract has been satisfied. This process is known as acceptance testing.(1:295)

This definition can apply to the acceptance of software both from a software developer under contract to your organization, as well as software which is developed in house. Regardless of the source of the software, there must be some method to determine if the software is performing correctly.

This problem of determining whether software is acceptable is complicated in the Department of Defense environment by the multitude of vendors who have contributed to a complete system. When hardware is being developed in conjunction with the software, the final software criteria may not be determined until the limitations of the hardware are manifested. Pressman states:

> The deliverable that is developed as part of the requirements analysis is the Software Requirements Specification. The specification extends scope by establishing a complete information description, a detailed functional description, appropriate validation criteria and other data pertinent to requirements.(16:116)

All classical design models have requirements analysis and specification as one of the earliest steps in the design process. The two do not seem compatible.

It is a well known assumption, called the 80/20 rule, that in recent years the total cost of systems has moved to 20% for the hardware and 80% for the software(9:22). Brannigan stated:

> As computer hardware has become much less expensive, it has become obvious that acceptance testing of the software is the critical stage.(1:295)

In an article concerning the legal problems involved with acceptance testing he says:

> In legal theory, there is no problem with acceptance testing. The contract specifies what tests are to be performed, the tests are run, and the software either passes or fails.(1:295)

But he goes on to state: "In practice that is not the case"(1:295). Often the buyer contracts for software and the vendor modifies an existing package and installs it. It has fulfilled the contractual requirements, but is nowhere close to what the buyer wanted.

Acceptance testing is complicated even further when the user is inexperienced in the use of the package he wishes to buy. In effect he is bargaining at a severe disadvantage. The vendor knows exactly what his product will do and what its capabilities are. He probably knows from reading the contract if he can meet the contract specifications before the testing ever begins. The buyer on the other hand, has

just decided that he cannot economically develop the software system in house and is looking for help. This is the same buyer who now has to write a contract and define the acceptance test criteria. One method is to:

> . . . form a series of inputs to the software and test the program's major requirements. All program statements should be tested at least once.(2:437)

Validation Testing

After the acceptance testing has taken place and the software is installed and in use, it is natural to expect that changes will need to be made by the software developer. "The problem is compounded because programmers constantly have to update software to keep it from becoming obsolete"(14:66). In most cases these changes will actually be requested by the user. The system that was originally defined is not likely to fulfill the user's needs forever. With every subsequent release of software, some type of acceptance testing needs to be initiated. This is referred to as validation testing.

> The reliability of software can play an important role in determining when a system should be released, whether it should be accepted by the user, and the degree of user satisfaction once the system is operational.(13:338)

This is true not only in the case of acceptance testing but also for validation testing. There are several different situations requiring the need for a new release and consequently a new validation. If the software is not

providing any new capabilities, the original acceptance test may be repeated as a good method to revalidate the package. If new paths are opened in the execution of the code, then these paths must be tested to revalidate the package.

"In software testing the main statistical emphasis is on estimating the number of errors remaining in the system"(13:338). But the main problem is not in determining the number, but in finding the errors before the system goes back into operation.

One of the best methods for validation testing is to use a third party to conduct the validation, just as a third party is the best method for original acceptance testing. But this can quickly become costly. To protect the user, software must be revalidated for the smallest change that affects the baseline of the system. Even the load of a different operating system or the use of a different compiler by the software developer, calls for a validation by the releasing authority. If a third party cannot be hired for each of these validations, what are the choices?

Two good strategies for validating the software involve the use of either a remote terminal emulator (RTE) or an internal test driver. The internal test driver is generally the tool of choice for in house software validation because the internal driver can run on a single machine and seldom requires additional resources. Software validation, especially path verification, can be done in a single

threaded mode which is ideal for an internal driver. In an editorial, Ralph Evans contends:

> Sequential tests are hailed as great savers of resources in reliability testing. . . . Rarely is a system rejected because its test path touches the reject line. What happens is that . . . the faults found in the test are fixed and the system is issued.(8:337)

Sequential testing and path traversals always have one large deficiency; they only execute the paths or code, one at a time. What is needed is the same type of a load as the operations system encounters. This is where the remote terminal emulator surfaces as a software validation tool. The RTE can collect an actual work situation and then when connected to another machine, resubmit these identical transactions to the system under test. The resubmission feed can occur at the original collection rate, slower or faster. In the case of a faster feed, the rate can be increased until the system under test either backlogs or has an overload fault in the software under test.

Summary

The purpose of this literature review was to present information in other sources pertaining to the testing of software systems, especially acceptance and validation testing. In order to fully understand why a remote terminal emulator can be such a useful tool to a software configuration manager, it is necessary to realize the broad scope of software testing. This chapter gives the reason for the implementation and evaluation chapters which follow.

III. Modification and Implementation

History

The remote terminal emulator package chosen for this implementation was originally designed by the Computer Sciences Corporation (CSC) under contract to the Computer Services Directorate of the Joint Data Systems Support Center (JDSSC). Support for the maintenance of the software was originally also supplied by CSC for the Computer Performance Evaluation Branch of JDSSC.

The original request for full screenface emulation was presented to JDSSC for their consideration. A preliminary attempt to transmit and receive full screenfaces occurred at Scott AFB, Ill. in December 1984. The installation of the RTE package under the direction of CSC used asynchronous communications channels. The evaluation of the implementation showed that characters of the transmissions were being lost with no indication that an error had occurred.

The contract for the maintenance and technical support of the RTE was subsequently awarded to Advanced Technology Systems of Vienna, Virginia, still under the direction of the Computer Performance Evaluation Branch of JDSSC. It was with their support and efforts that the modification necessary for this implementation took place.

Three distinct areas had to be addressed to make this implementation successful: 1. the RTE software had to be brought up to the current operation system used by the sponsor's machines, 2. the RTE application software had to be modified for full screenface stimulus and response as well as full screenface response verification, and 3. the physical connections had to be accomplished with cables instead of through a patch panel. The order of discussion of these three entities in this chapter is not intended to order their importance. If any of the areas had failed, the result would have been a failed implementation.

Modifications Due to Systems Software

Considerable effort and examination was originally directed to the difference in the operating systems installed on the hardware systems. The DPS-6 Model-95's at Scott AFB, run on a commercially procured operating system. The currently supplied version of this operating system is labeled as version 3.1. In addition to being the most current operating system available, the operating systems on these machines are updated quarterly with an update package supplied by the vendor. These update packages include the latest modifications and corrections that are available. The contractor supporting the RTE software is supplied computer time on two different systems at the Deery Engineering Building in Reston, Virginia. One of the machines is a Level-6 Model-43, which was running under a Worldwide

Military Command and Control System (WWMCCS) released operating system. This is a version of Honeywell's operating system 2.1. The other machine is a DPS-6 Model-95, which also was running operating system version 2.1.

The sponsor's experience in converting their COBOL applications from version 2.1 to version 3.0 led them to believe that the conversion of the RTE software would be a long and arduous task. It was not. The only problem encountered was on an early visit to the Reston, Virginia site, when the Polled VIP Emulator software was not yet in place on the DPS-6 machine. No problems attributed to the operating system were encountered when transporting the software to Scott AFB. The implementation would have been easier if the SUT application software could have been loaded at the development site, but because the application software runs the commercial 3.1 operating system, it could not be loaded on the development machine to test prior to implementation.

Because the implementation of the RTE was on systems used daily by the sponsor, it was necessary to convince their users that there would be no impact or disruption of their normal service. The solution was to totally isolate the RTE test from the users. Block time was scheduled on the machines for the RTE test team's exclusive use. To complete the isolation, both the operating system and the SUT application software was transferred to a removable pack. A

smaller version of the data bases needed by the application software was duplicated on a second removable pack. This allowed the RTE application software to be loaded at implementation time without fear of disrupting normal operations. At the beginning of test block time, the machine was booted off of the removable packs and rebooted off of the normal system packs at the end of the test time. During the test time, the user's packs were write protected.

A release of the SUT application software and data bases occurred during the months of testing. The disk packs we created were not upgraded to the new software. As noted in Chapter 1, the intent was to evaluate the RTE as a tool, not do actual testing for the sponsor.

The actual implementation at Scott AFB required the changing of the file used to cold boot the system. No attempt will be made to explain the changes necessary to the file or to explain how the software was actually loaded. The RTE Configuration Manual (6), devotes the first 32 pages to these changes. Several of the operating system manuals are required to understand the approach in addition to the configuration manual. They are: 1. System Building and Administration (12), 2. System Concepts (10) and 3. System Messages.(11)

Modifications to RTE Software

After the initial test of the full screenface transmission in December 1984, several changes were made to

the software in order to implement the full screenface mode. These changes were all targeted toward implementation on asynchronous lines. The original contractor was not able to deliver a successful full screenface product to the sponsor.

This implementation was aimed at implementing synchronous terminals. There were two reasons for the synchronous choice. First, if a collection of actual data at an operational site is contemplated, the RTE software must be able to collect and retransmit on the types of devices in the field. Most of the operational sites of the sponsor have both PACS and CARGO terminals running on synchronous lines. Second, the contractor felt that the ability to detect the completion of a transmission on a synchronous line would help solve the problem of lost characters.

After the move to synchronous lines, there were two key problems to solve in order to implement the full screenface package. The first involved the sizes of the stimuli and responses; both were limited to 72 characters in the original package. The second was the ability of the RTE software to verify on five characters of the response against a previously collected response. In solving each of these problems, another problem was generated.

Expanding the size of the stimuli from 72 characters to full screenfaces could have drastically changed the size of the bound units. When adjusting the size of the stimulus and response from 72 to 1968 characters, the number of terminals

to be emulated was dropped from a possible 128 to 4. The size of ASCII text for the stimulus alone for 128 terminals would have been 245,760 (128 times 1920). An equal amount of space was needed for the responses.

By dropping the number of terminals during the emulation phase, the sizes of the buffers to support those lines was kept to a reasonable limit. In its emulation phase, the RTE uses 120K words of memory to perform an emulation on four terminals. It is likely that the sponsor will ultimately request or build two different versions of the RTE emulation module. One for software validation, where four terminals would be sufficient, and a larger module needed to emulate additional terminals for stress testing. For software validation a single hardware system approach would be used and memory would be limited. For stress testing, the RTE application software would reside on its own hardware system and memory would not be a problem. Each of these concepts will be addressed in Chapter 4 (Software Validation) and Chapter 5 (Stress Testing).

The second problem area was expanding the response verification from 5 to a maximum of 1920 text characters. The original package was limited to 5 because the speed of the emulation was severely degraded when the length of the comparison was increased. This was due in part to the way Fortran generates its storage blocks on the DPS-6 at execution time and in part to the speed of the memory to

memory compare. The sponsor determined total response verification was a mandatory requirement and that during the software validation phase of testing, speed was not a critical component. If verification is needed in conjunction with stress testing, the verification module may need to be converted to the assembler language level.

Hardware Implementation Details

The physical cabling on the RTE implemented machines was accomplished at Scott AFB, without any previous tests. The channel ports on the Level-6 machine at the contractor's development site are wired directly into a patch panel. This panel connection can then be routed to a terminal or a port on another machine. The sponsor does not have a patch panel to connect the two systems. The ports on the sponsor's machine were directly connected.

Two cables were necessary to connect the sponsor's DPS-6's. The most important cable was manufactured during the initial software testing at Reston, Virginia. This cable transfers the stimulus data from the RTE implemented machine to the SUT. It also accepts the response from the SUT on the return lines. Figure 4 shows the suggested diagram for wiring a cable to be used with a patch panel. Figure 5 shows how the direct connected cable was constructed. During the initial testing through the patch panel at the development site, it appeared that the timing signal was not strong enough to drive three lines tied together. In Figure 5 there is no line that drives more than two.

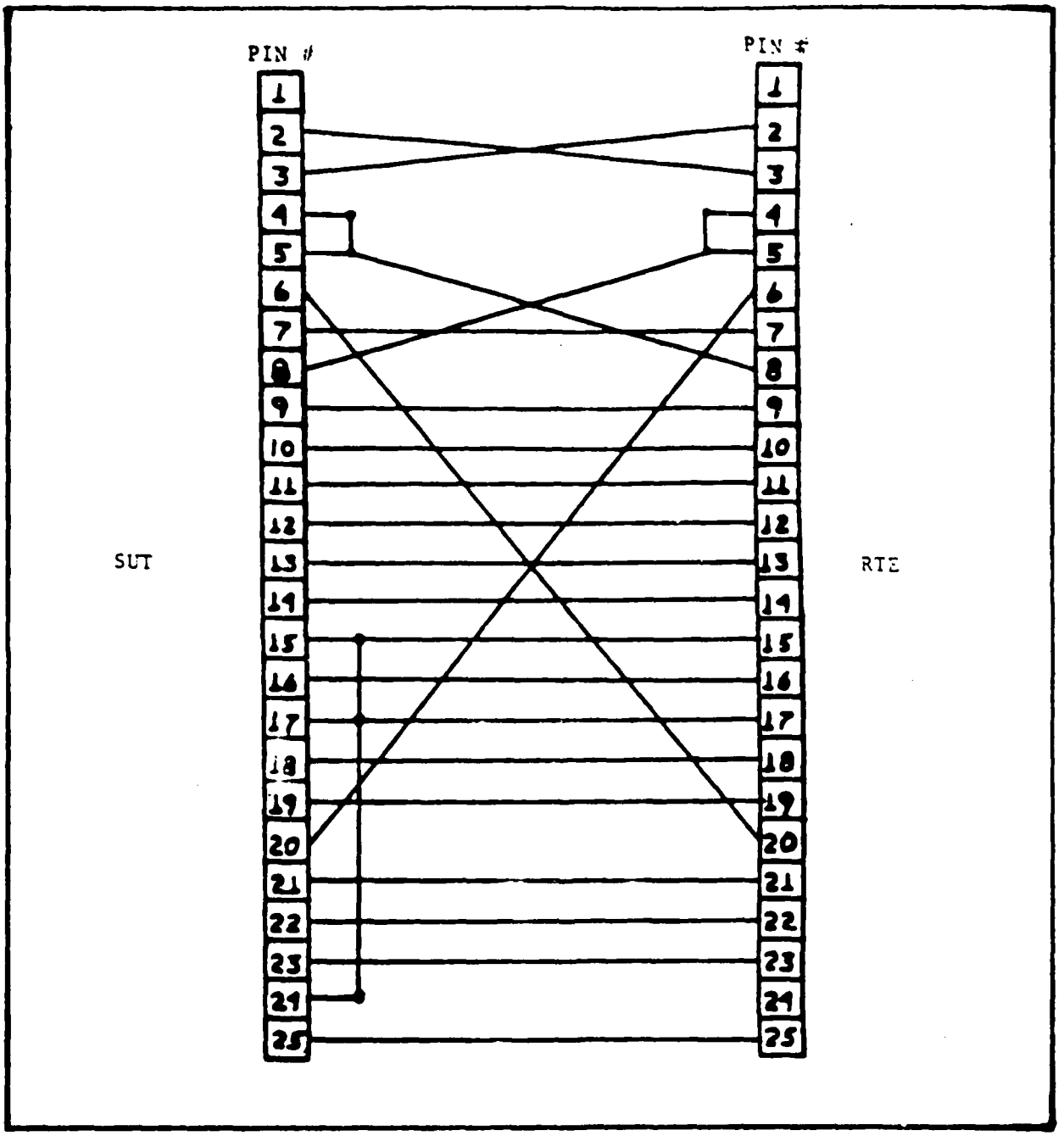


Figure 4. Patch Cable Schematic (6:2-148)

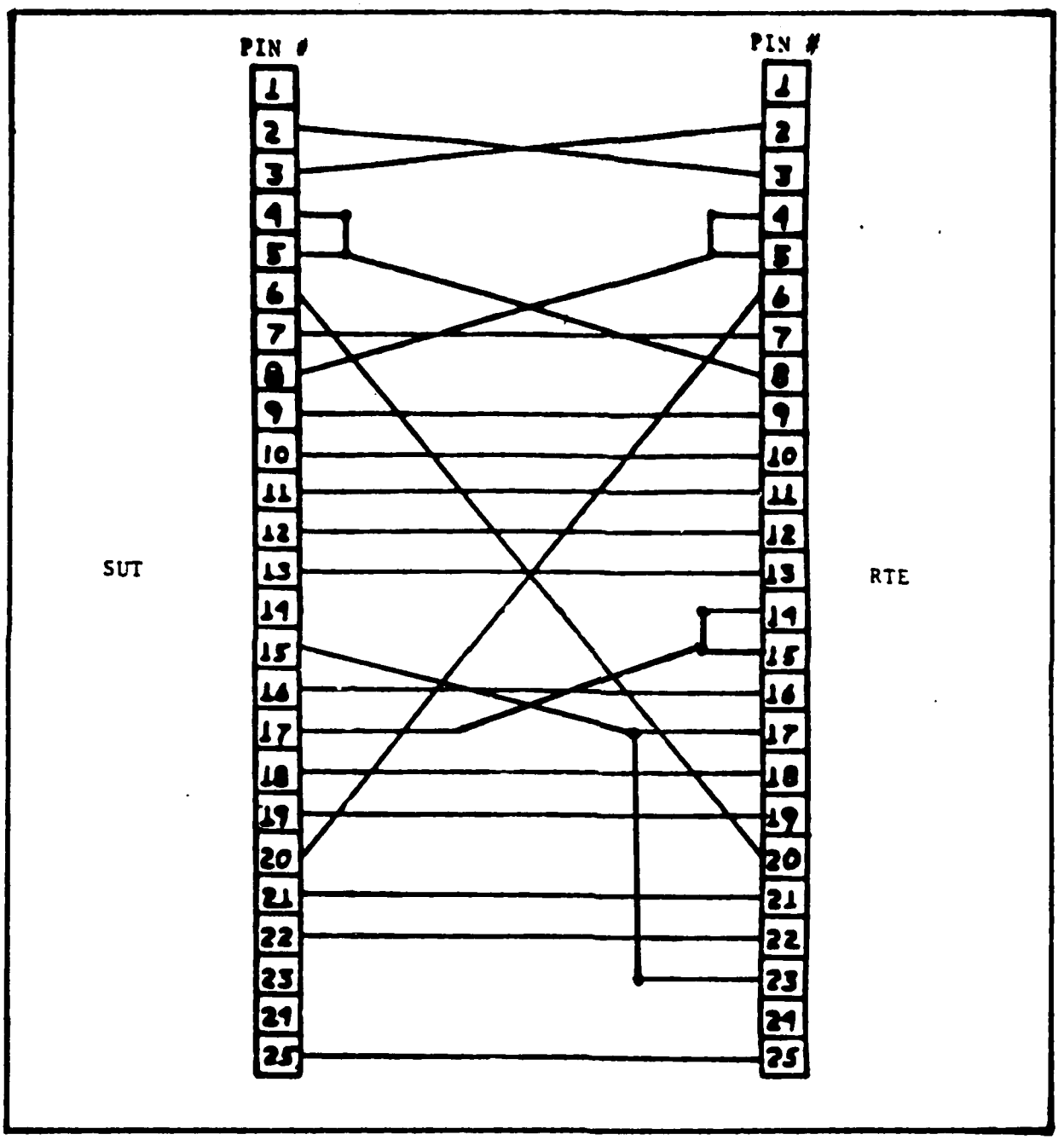


Figure 5. Revised Crossover Cable Schematic

Up to 32 different terminals could be emulated with this cable and two ports on the DPS-6 if the RTE emulation software was expanded beyond four terminals. During the implementation, the machine was configured with four logical terminals on a single physical line. The Configuration Line Manager (CLM) User's file entries to support two lines are shown in Figure 6. The stimulus flows on the PVE01 line to the STD04 line and the response from the SUT application software returns on the STD04 to the PVE01. The SUT is unaware that it is being fed by another line instead of an actual terminal.

It is immaterial whether the PVE lines are configured on the SUT or another DPS-6 machine. In the initial implementation, both ports were configured on the same machine. This was to allow the other machine to be used by the sponsor. For the stress test evaluation, the RTE software and the PVE lines were moved to the second DPS-6. This required a second cable. A 25 foot cable with RS-232 connections was used to connect the PVE port of the RTE machine to the STD port of the second machine. In order to include the crossover cable of Figure 5, it became necessary to continuously route the signals through a data scope. Both cables had male ends and a coupler with two female ends could not be found. For future work, either a coupling cable or another 25 foot cable with different ends should be found.

```
PVE 201,10,X'C900',0,0,QA          TEST ENTRY
DEVICE PVE01,201,10,X'C900',,2500
PVE 202,10,X'C900',0,1,QA          TEST ENTRY
DEVICE PVE02,202,10,X'C900',,2500
*PVE 203,10,X'C900',2,2,QA          TEST ENTRY
*DEVICE PVE03,203,10,X'C900',,1970
STDLN 10,X'C980',0,2400,W4         TEST ENTRY
STD 204,0,,'7814S',PB                   TEST ENTRY
STD 205,1,,'7814S',PB                   TEST ENTRY
*STD 206,2,,V7700                   TEST ENTRY
POLIST 1                           TEST ENTRY
STAPOL 0,1
DEVICE STD04,204,10,X'C980',,2500       TEST ENTRY
DEVICE STD05,205,10,X'C980',,2500       TEST ENTRY
```

Figure 6. CLM User File Entries For Two Terminals

A significant implementation factor was the ability of the sponsor's personnel to attach the cables and hardware to the physical machine. During the period of the RTE test, all test personnel had physical access to the cables and ports. At the Reston, Virginia development site, the test director had to request cabling support from the field engineer which often took considerable time. At the sponsor's site, the DPS-6's are located across the base from where the field engineers work, which would increase the time needed to recable. If the RTE was to be used daily, it would be reasonable to have at least the STD port on the SUT permanently cabled.

Testing

After loading the RTE application software, rebooting the system and cabling the ports, the actual implementation was examined. The testing was divided into two phases: 1. checking out the physical hardware and 2. evaluating the RTE software.

The physical connections were evaluated using a DLM IV Data Line Monitor. The DLM IV was very easy to use and the operator's manual (7) was well written. The disadvantage of the DLM IV was that it had no capability to print the collection. An even better package for the sponsor to examine is a utility which tracks data coming across the channels. This utility was available for previous versions of the operating system. Several loose connections caused problems during the initial evaluation. It is important to not only plug in the connections but to tighten the mounting screws, leading to the recommendation that the port attachment on the SUT be permanent.

The evaluation of the RTE software progressed from simple to more complicated. It ranged from a stimulus of 10 characters ($LOG-ON;CI) and a response of 14 characters (INVALID LOG-ON), to a full screenface input and output. No serious errors were noted in the way the system was configured to run, but several minor modifications of the CLM file were required. Two errors were noted in the collection and building of the data base necessary for an emulation

session. Both concerned a single precision counter in a Fortran program. As the character count exceeded 32K the counter reset and data was lost. Both programs were later modified to eliminate this problem.

Summary

It must be noted that the sponsor acted as a test site for the software developers. Most of the software was specifically written to help the sponsor in his test package. In the initial evaluation of the package, several minor problems were found. These problems were quickly corrected, sometimes patched within the hour. This initial package performed much better than the sponsor had anticipated.

IV. Evaluation as a Software Validation Tool

Introduction

Chapter III described the modification and implementation of the RTE on the sponsor's machines. The final stage of the implementation insured that the RTE application software and the hardware connections could function as indicated in the accompanying documentation. The second phase of this thesis was to evaluate the RTE as a software validation tool. In other words, how well could the RTE software determine if an application program or system under test, was returning an erroneous response to a predetermined stimulus? For the purpose of this effort, response verification is defined as a character by character compare of the original response versus the emulation response.

To be considered a viable software validation tool, the RTE had to adequately perform three different functions. They were: 1. collect stimuli and responses going to and from the SUT application software, 2. combine those collections into matched sets, and 3. retransmit the stimuli and verify the responses from the SUT to the ones previously collected. The sponsor was interested in not only evaluating the RTE as a software validation tool, but also in comparing its use to that of his current internal test driver (ITD).

Consequently, the focus of this chapter is examining how well the RTE does its primary task of validation. If it is useful in software validation, does it do it better or faster than the internal test driver? Is it easier to learn and use?

In order to insure comparability of the data, the same stimuli and responses were used for both the RTE and the ITD. The terms inputs and outputs will be substituted for stimuli and responses occasionally. They refer to the same transactions. Also, in order to insure no compromise of official passenger data if or when the results were turned over to the RTE contractor, all social security numbers and names are fictitious. The passenger system data bases contained no actual data. Examples of the inputs and outputs are found in Appendix L. They are printed in a format generated by the ITD.

Stimulus and Response Collection

The input-output collection of the RTE package is done by a program called Autogen. Figure 7 shows how the RTE collects inputs and outputs. At initialization time, the Autogen program asks the user which line the inputs will be sent out on. PVE01 is used in the diagram. After this PVE line has been entered, the user establishes the STD line as a valid terminal. In the diagram it is STD04. At this point, any system log-on message is sent out the STD line, through the PVE line, received by Autogen, written to the response

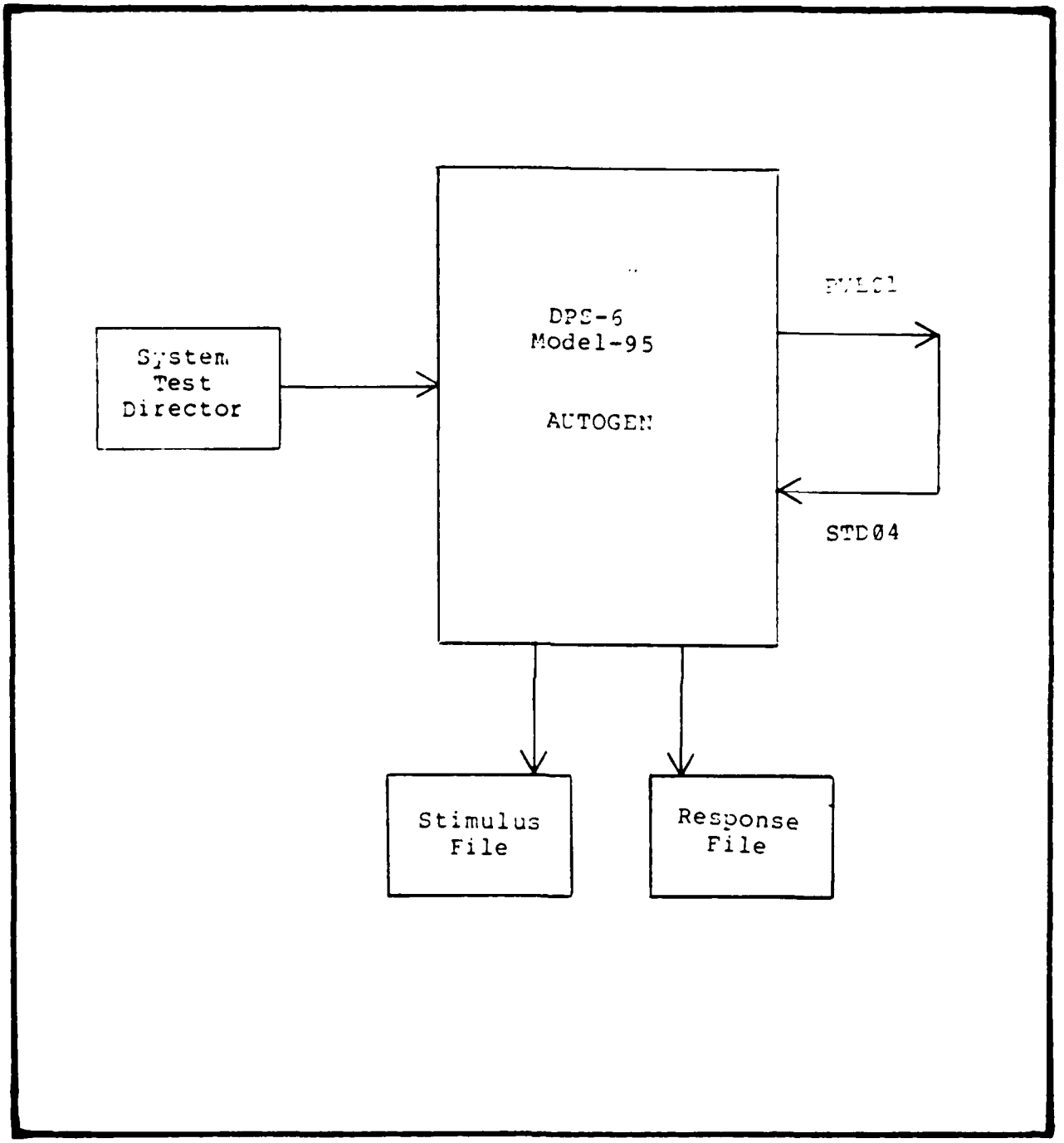


Figure 7. Diagram of Remote Terminal Emulator's Data Flow

file as response number zero, and transferred to the original terminal. The connection is now complete and any inputs entered on the original terminal are logged by Autogen in the stimulus file and processed by the SUT after being passed to STDØ4. Responses are collected in the return circuit. Examples of the stimulus and response files are found in Appendix C.

The collection software is easy to use. The user needs to be told, by the configuration manager, what line the RTE uses and what line to bring up as a system terminal. The RTE user enters his own name for the collection files when prompted by Autogen. At that point, he is unaware that the collection is happening. He ends the collection by transmitting '&&&&'. '&&&&' was chosen by the RTE's initial developer as a terminating character string, because it was not expected to occur in a normal input string. At that time, the Autogen software acknowledges the end of a collection session. Appendix C gives a file dump of both stimulus and response files for a part of a collection.

The evaluation of the RTE collection program found it to be easy to use and totally accurate in its collection. Due to its double buffered files, there was no noticable delays when receiving or sending screenfaces. The Autogen program required only 67 kilobytes of memory on a 2 megabyte machine. The only noticable drawback was the file space necessary to store the files. If a number of terminals were

being collected in an operational environment, available file space would be quickly exhausted.

The sponsor's internal test driver uses a computer performance evaluation package to log inputs and outputs to a journal tape. It collects data as it is transferred to and received from the input-output handlers. Previous studies had shown a five to six percent degradation of response times when a collection is occurring. In the case of the ITD, the user is totally unaware of a collection. Therefore, there is no real difference to the user in the collection of the RTE or the ITD.

Script Assembly

The packaging of the inputs and outputs into matched pairs that can be retransmitted to the SUT is referred to as building and assembling a script by the RTE manual and labeled scenario management by the ITD and the sponsor. Regardless of the terminology, the aim is to generate a continuous combination of inputs and outputs to pass to the SUT.

The RTE uses two different modules to generate the script for the emulation phase: 1. the Build function of Autogen, and 2. the module called ASMB. Both are menu driven, or question and answer oriented and very easy to use. It is at this point that the RTE and the ITD begin to differ drastically.

The Build function of Autogen provides a level of capability not supplied by the ITD. This level is the ability of the RTE to reprocess transactions at the speed in which they were collected. In other words, if the user waited 20 seconds between the receipt of a response and the transmission of the next stimulus, then the RTE collects that wait time and makes it available for the retransmission. In some literature, this is referred to as think time. It is in the Build function that the RTE allows the system test director to specify either a default wait time, or an actual wait time.

If actual wait times are not selected, then the default wait time is used. Consequently, to run an emulation without wait times requires either eliminating the wait times and changing the default time to zero or changing all of the wait times to zero during the build function. This is a very tedious procedure, especially for a long scenario. The test director found it much easier to build the scenario file, then read the file into an editor and globally change all the wait times to zero. An example of the completed script file is in Appendix C. While this function supplies an option not available in the ITD, the RTE process is slow and monotonous. Some type of global change should be supplied in future releases.

It is also in the Build function of Autogen that the groundwork is laid for response verification. This is a

function that the ITD supplies automatically, since the ITD was written specifically for the purpose of response validation. It should be noted at the beginning of this critique that the contractor already plans a modification allowing global response verification much like the ITD of the sponsor.

The method to establish response verification is very cumbersome. The actual screenfaces for a single transaction are included in Appendix D. After specifying either default or actual wait time, the Autogen software displays stimulus number one on the screen. At this command level, the user enters 'R' to signify response verification. Autogen displays the response on the terminal and asks if the starting point for the verification is on that page. The RTE only displays 9 of the 24 lines per screenface. After arriving at the correct screen, it prompts for the desired starting line. To establish the actual response verification starting point, it displays an 80-column scale and prompts for the starting column. Autogen reiterates this procedure to find the ending test line and column. It terminates the building of one response verification transaction by reprinting the original stimulus information plus the response to be verified.

This method of specifying the response is very slow. At a minimum, it takes six screenfaces and eight responses by the user to flag one transaction for verification. Even a

300 input scenario would take longer than this writer would care to spend. It is in this utility that the RTE lags far behind the sponsor's ITD. Perhaps the future release with global response verification will eliminate this problem.

To prepare for the actual emulation session, both the RTE and the ITD use a simple utility. To assemble a script in the RTE, the user enters the utility ASMB which queries the user for the data base name, a name given to the actual prepared script, and the script name from the previous Build function. The results are data base files needing only the appropriate line and channel numbers. The ITD assembles its script by querying the user for a start transaction number. It then strips the journal tape matching the inputs and outputs in an indexed file. Both systems are well documented, easy to use, build about the same size of files and execute at roughly the same speed.

One severe disadvantage of the RTE is its lack of utilities to merge scripts and to enter individual stimuli and responses after the script has been completed. There is no reasonable way to enter a single transaction in the middle of a previous script, other than building a script of one transaction. With the one transaction script, the user could divide the original into two scripts at the point of addition, then combine all three back into a single script. The sponsor's ITD allows additions at any point and the RTE software needs this modification to improve its usefulness.

Emulation

Evaluating how well the RTE does its primary job of response verification or software validation is best done by walking through the steps to begin an emulation session and comparing the results with the sponsor's ITD. To begin a session, the scripts must be prepared as described earlier in this chapter and the lines must be configured and matched to the scripts in a utility called Config. To begin an emulation session, the test director (user of the RTE) enters the utility called RTE. RTE queries the user for the data base (script) and then asks the user how many iterations of the script he wishes to run. This is a most useful tool and will be discussed in more detail in Chapter V. The only direct need in software validation, for this capability, is if the test director is looking for an error that occurs after an extended period of time. Our test scenario was designed for repeatability and a test was conducted with five iterations with response verification. The ITD has no corresponding capability and while it is not a primary need for software validation, it is just the thing that may enable a system test director to find a tricky time-caused error.

The RTE next asks for the number of errors allowed in the script emulation. If the RTE discovers an error, it means that a response from the SUT is not identical to the previously collected response in the data base. It then counts the number of errors that have occurred so far in the

emulation and examines to see if the total has been reached. If so, it ends the emulation. If not, it writes the incorrect response to a report file (XLOG) and continues the emulation.

This allows the system test director to test a scenario and examine the errors after the completion of the test. The ITD stops a scenario as soon as it finds the first error and transmits the incorrect response to the terminal of the test director. It then waits for the test director to provide instructions to continue, stop, or repeat the offending stimulus.

Which system is the better is really a matter of choice. The RTE does allow a hands off approach for software validation, but the sponsor found that with stimuli that change the data base, if one response is wrong, chances are that all subsequent responses will also be wrong. Therefore, for the sponsor at least, this would not be a strong advantage.

The third step in emulating the script is to start the actual transmissions. This is the area where the RTE surpasses the ITD. Providing the scripts have been collected correctly, the individual lines can be used to exercise different application programs at the same time. The ITD is limited to supplying one stimulus and waiting for the corresponding response. It can then send another stimulus which could be targeted toward a different application

program. The RTE can execute four different application programs at the same time. This can easily uncover data base contention errors or other errors caused by resource contention.

The RTE allows the test director to actually exercise the communication line handlers used by the system under test. The ITD is totally within the system and does no communication with the terminals or terminal drivers. For the sponsor, this is a large concern since the input-output handlers are user written. The RTE also allows the test director to perform computer performance evaluation on the actual transmissions.

Summary

The RTE surpasses the ITD as a software validation tool. While the RTE is not as versatile in scenario management as the sponsor's ITD, the addition of a global response verification capability would make the RTE just as easy to use as the ITD. The RTE is extremely easy to operate. It does a better job of software validation than any tool the sponsor currently has. It not only exercises the SUT application software, but also exercises the communication channels and the communication oriented application software.

V. Evaluation as a Stress Testing Tool

Introduction

Chapter IV examines the use of a RTE as a software validation tool, by evaluating how well it is able to verify the correctness of responses which are being returned by the system under test. This type of software validation testing is most often done in a very controlled environment, often with predetermined scenarios which were constructed from an acceptance testing plan. This thesis effort continued beyond the evaluation of the RTE as a primary software validation tool. Chapter V evaluates the RTE as a stress testing tool.

The presentation of this chapter is divided into four areas. They are: 1. the concepts of RTE stress testing and how it is needed by the sponsor, 2. the method of collection for multiple terminal emulation, 3. the emulation of multiple terminals on a single machine, including results of sample runs, and the emulation of terminals from one machine to another, and 4. the problems uncovered with the RTE during the sample runs.

Concepts of Stress Testing

To be considered as a viable stress testing tool, the RTE must be able to feed stimuli and receive responses from the SUT in such a manner as to approximate different levels of use or a different number of users. The sponsor found that software which was validated by the internal test driver

would often fail after being loaded at an operational test site. These failures were often due to the interaction of application software after it had been placed under a specific load of users. These load related failures were extremely difficult to diagnose because the operational sites have no on-site programmers or analysts. In addition, the load could not be duplicated on the sponsor's machines. The ITD was capable of only single-threaded input and the sponsor does not have enough terminals to generate the load, even if he could find knowledgeable users. To further complicate the problem, the software failure often occurred only with a precise mix of transactions. The result was generally a programming team sent to the site.

The capability to emulate multiple terminals is the area where the RTE would far surpass the ITD. Emulation of multiple terminals would allow the sponsor to load test the new software to determine if it had stress related errors. In addition to the software test under load, the RTE would allow the system configuration manager to test the hardware configuration at each site to determine if it had the processing power to insure adequate user response times. Coupled with the sponsor's computer performance evaluation subsystem, stress testing the SUT would allow the manager to evaluate the effect of additional users and additional loads on the SUT. This need for capacity planning data is a natural by-product of the stress testing phase.

The RTE also would allow another avenue of testing, never before used by the sponsor. Each operational site generally contains at least two sets of hardware. As described in Chapter I, the sponsor runs three different operational systems on the two sets of hardware. PACS and EARLO usually run on one hardware set with CARGO on the other. In the case of a catastrophic hardware failure which disables one machine, the data base from one system is transferred to the other. This is possible because the data base is stored on removable packs. In the cases of a disk crash, the previous save is used and a journal tape recovery brings it up to date. Application software for all three systems is always current on both sets of hardware. By transferring selected physical terminal lines, the backup configuration allows both systems to run in a degraded mode. After collecting scenarios on a PACS and CARGO system for software validation, the RTE can be used to simultaneously feed PACS and CARGO stimuli to the SUT on different logical lines, thereby evaluating the feasibility of a backup configuration.

Collection for Stress Testing

The type of testing, or more importantly the focus of testing, will generally determine the method for the collection of stimuli and responses. Chapter IV discussed in detail the procedures to capture the stimuli and responses going from one terminal into the SUT. If the stimuli and

responses are a repeatable scenario, then the collection from a single terminal is a valid collection method. An example of this type of collection, would be a normal user's session from a software developer's terminal. These types of stimuli, such as compiles, editor work, lists of directors, etc., are generally repeatable regardless of the order. Because they give similar results, a single session can be collected and repeated over several terminals. Chapter IV described how the single script could be assembled into a data base for emulation over several lines. However, even in the simple case just described, things can go wrong. A compiler invoked from two different terminals, using the same source file, may each try to establish an object file with the same name, or a source listing file with the same name. This leads to a file contention problem during emulation.

While the test scenario was repeatable, it was designed and constructed that way specifically for testing the RTE package. Part of the intent of this chapter, is to show that the RTE is capable of multiple collection from different terminals. Figure 8, when compared to Figure 7, shows the additions necessary to collect stimuli and responses from two different terminals. The collection in this study was limited to two different terminals. This was due to a hardware limitation on the sponsors machine. Both DPS-6's available for this study had only one synchronous terminal. Since the current version of the RTE was limited to

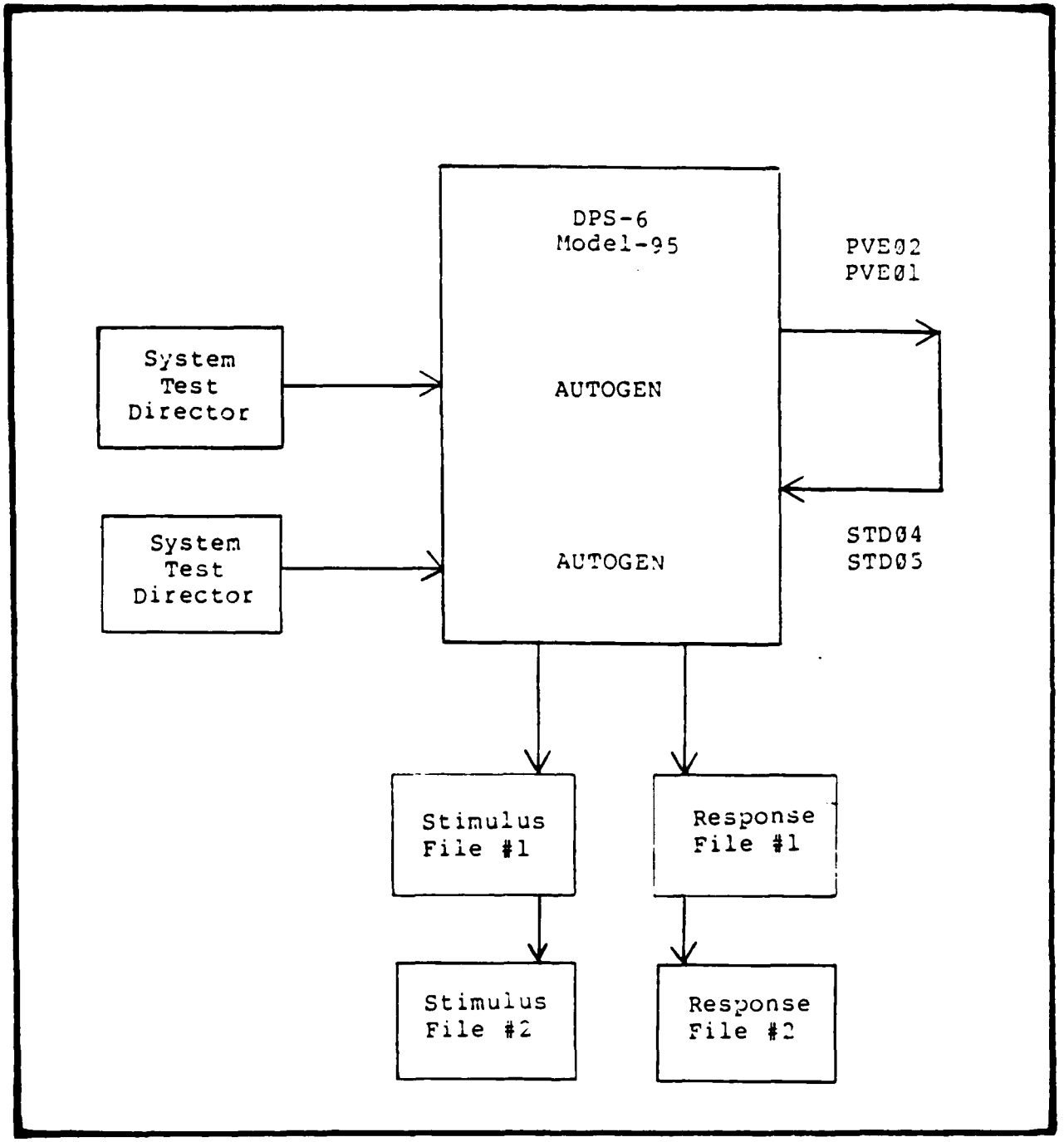


Figure 8. Diagram of Data Flow for Two Terminals

synchronous terminals, one asynchronous terminal was converted to a synchronous mode. This required changes to the CLM user's file, recabling to a synchronous channel, and resetting of internal switches on the terminal. After configuring the second synchronous terminal, the collection was accomplished as diagrammed in Figure 8.

An important point of this diagram is that the diagram shows only one physical line going from the PVE to STD port. In the test, the first control terminal was directed out of PVE01 and into STD04; the second out of PVE02 and into STD05. The DPS-6 is able to handle 32 different logical lines on a physical line. To further test the RTE stress test collection capability, two synchronous test director's terminals and four asynchronous terminals were configured through Autogen. The system had no trouble handling the physical requirements, but a collection through synchronous lines from an asynchronous terminal is not possible.

Each Autogen module requires 67 kilobytes of memory, so a potential memory problem exists at an operational site if all terminals are used for a collection. Most of the operational systems do have 2 megabytes of memory to cover the backup configuration requirement, so a reasonable choice of terminals should give adequate coverage. A larger problem might be the funnelling of all the logical lines into a single physical connection. The sponsor does have 32 hand-held terminals, routed through a micro-computer into a

single physical line. They have noticed severe degradation of response times when traffic from the hand-held terminals is high. The same situation would likely occur with a concentration of terminals going into a single channel.

It was the intent of this study to evaluate the feasibility of the stress test collection, and to determine if the RTE application software could handle the collection. The RTE software was capable of adequate collection from more than one terminal. Because two machines are available at most sites, it was also suggested that both machines could be utilized as in the emulation phase. This isolation of the RTE software on the other machine also worked without error. In summary, while the entire package performed without error, before traveling to an operational site for a full site collection, considerable preparation would be required.

Stress Test Emulation

Often the effort the system test director can devote to testing is limited by the resources that he has available to him. For this reason, even the stress testing of software would be broken into two distinct phases. First, the new application software would be driven with the RTE software on the same hardware systems and second, a final test would be run with the RTE software split from the SUT. This single machine test allows the test director to discover most of the load errors without the use of both machines; the two machine test allows a full and final test with computer performance evaluation data being generated.

To evaluate the stress testing capability of the RTE on a single machine, four separate runs were conducted. Each run used the same repeatable scenario initially generated during the software validation testing. As stated earlier in Chapter IV, the intent of this study was to evaluate the use of the tool, not to do actual testing. The sponsor's application software for the SUT is single-threaded. For this study, single-threaded is defined as being able to handle only one input at a time. In other words, they are not reentrant. The RTE software needed to be able to handle the situation where software is queued, awaiting entry into the application program.

Full use was made of the sponsor's Computer Performance Evaluation (CPE) subsystem, and it was with the CPE data that a major flaw was discovered in the RTE package. This flaw, a time delay problem, is discussed later in the chapter. Appendix E gives an example of the RTE report generated by a utility called XLOG and the corresponding CPE printouts.

Table 1 gives the condensed data for three of the four runs as generated by the CPE subsystem. To understand the data in Table 1 requires a knowledge of how the sponsor's CPE subsystem collects and generates the data. Figure 9 gives a graphic view of the CPE data collection. Three different record types can be collected for each stimulus and response sent and received by a terminal. Each of these record types has two time stamps, one indicating the start of the transaction and one at the completion. Type 10 records are

Table 1. Results from Stress Test Emulation

Test	Application	Total Stimuli	Number Queued	Percent Queued	Average for Queued	Average for Total
4-Line	IA	77	21	27	6.275	1.711
	IC	9	1	11	11.292	1.254
	ID	44	11	25	7.816	1.954
3-Line	IA	58	8	13	3.227	0.445
	IC	7	0	0	-	-
	ID	33	7	21	3.932	0.834
2-Line	IA	42	0	0	-	-
	IC	5	0	0	-	-
	ID	22	0	0	-	-

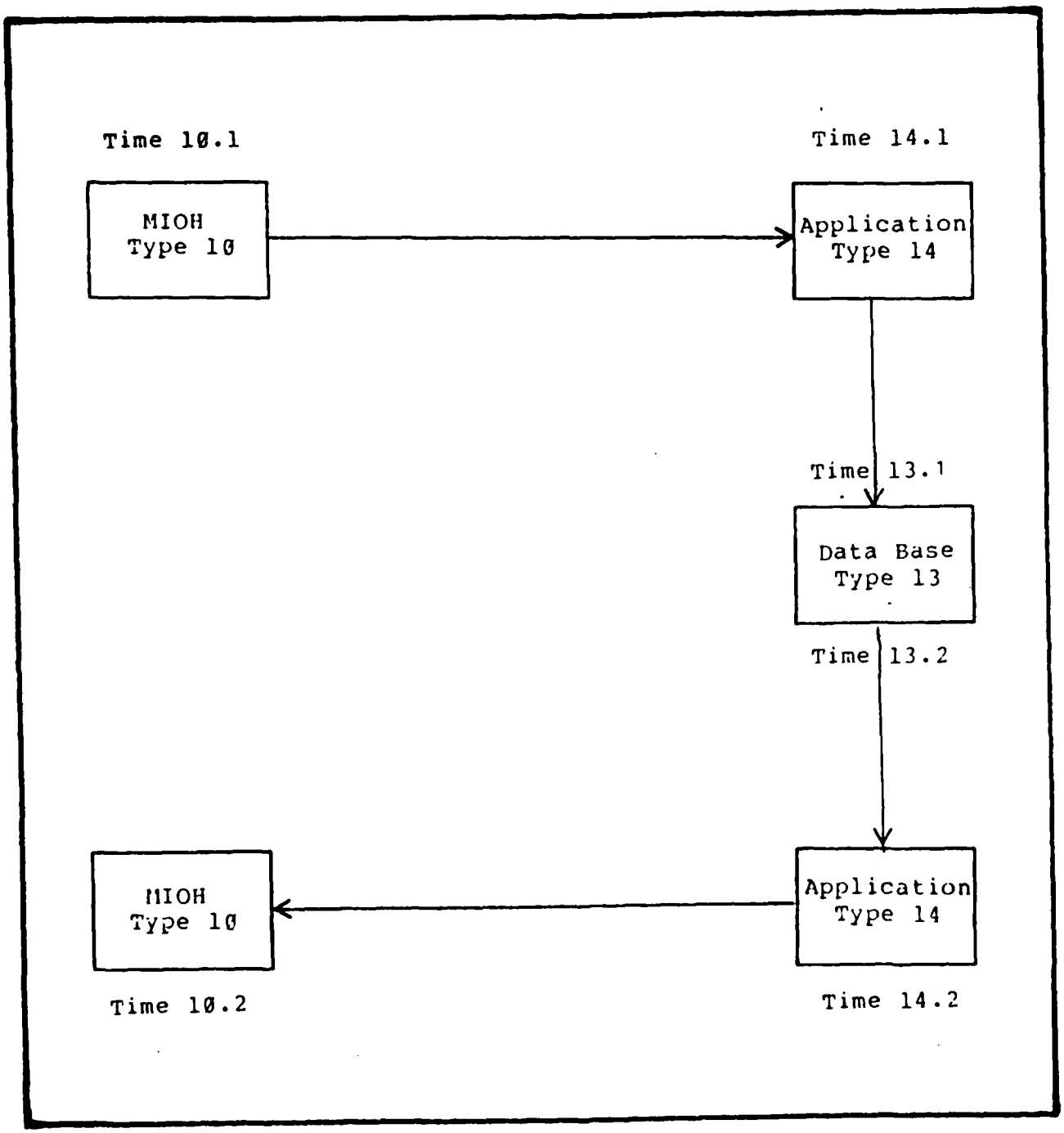


Figure 9. Diagram of Sponsor's Computer Performance Evaluation Subsystem

collected from the input-output handlers. Type 14 records are collected inside the individual application programs and type 13 records are generated by the data base system used by the application software. For the purpose of this study, type 13 records are not needed and will not be explained. The other two records will be described by explaining when the time stamps are taken in relation to the collection.

When the input-output handler receives the last character from the terminal, it generates a request for system time. The time is held and later journalized as time 10.1. It is important to note that the system time call is generated at the completion of the transmission from the terminal. The input-output handler releases the transmission to the appropriate application program, which issues a system time call that is later journalized as time 14.1. Upon completion of the application task, time 14.2 is generated, a type 14 record containing time 14.1 and 14.2 is journalized, and a response is given to the input-output handler for transmission to the terminal. The input-output handler requests a physical output to the terminal, waits for the completion of that output, generates a system time call, and writes the type 10 journal record with time 10.1 and 10.2.

In summary, a single type 10 and type 14 record is generated for each stimulus and response received from and sent to the terminal. The type 10 record includes one half of the terminal communication time, that half being the

output of the terminal screenface. The type 14 record gives the application service time for the user's request.

Because the input-output handler module is reentrant, with a separate buffer for each terminal, it is capable of serving multiple terminals simultaneously. The application program modules, on the other hand, are not reentrant and must serve the screenface requests in a single-threaded mode. After receiving a stimulus from the terminal, it completes all of the requested work, issuing an output screen for that terminal, before accepting the next input.

If requests for the application program arrive faster than the application can service them, then the requests are queued. To determine if a stimulus has been queued, involves comparing the time 10.1 of a record, with the time 14.2 of the previous record for the same application. If the 10.1 is lower than the 14.2, it means that a second or subsequent stimulus arrived before the application program was free to process it. If a stimulus must wait in a queue for the application program, then the time between record time 10.1 and record time 14.1 is a good estimate of queue time.

To be considered as a viable stress testing tool, the RTE must be able to transmit stimuli and receive responses faster than the SUT application software can generate the response screenfaces. In other words, it must be able to generate queued transactions. Table 1 does not include data for the single line transmission. At the single line level,

the RTE acts much like the sponsor's ITD, transmitting the next stimulus only after receipt of a response. There is no possibility of a queued transaction. The table results also show that a two line transmission does not generate queued transactions. The three and four line transmissions both generated queued transactions, showing that even with a simple test scenario, and with the limited number of lines available in this version of the RTE software, the RTE can be used as a stress testing tool. The four line test generated a queue for each of the application programs at least once, with over 25% of the transactions being queued on the two larger units.

It is noteworthy that with only four lines, the RTE generated an average queue time on the two programs of over six seconds for a stimulus that entered a queue. With a larger number of lines available in future releases, the sponsor will be able to stress test the SUT at will. By using a larger number of terminals, and adjusting the user think time between stimuli, he can achieve any level of use he desires to simulate.

Because of the limited availability of both machines at the same time, most of the data collection was done on a single machine. Additional runs were conducted, with one set of hardware as the RTE system, and the other as the SUT containing the application programs. There was no significant difference in the results of the one or two

machine runs. This is partly because response verification was not included as part of the stress test, and therefore, the RTE software did not consume significant CPU time. The single machine results were presented in the table due to the timing problem discussed in the next section.

Problems Encountered

There were three significant problems identified during the stress testing. They are presented in this section based on their level of importance in affecting the sponsor's stress testing capability. Where possible, potential solutions are offered.

As stated in the previous section, a timing problem exists when using two different machines to conduct a stress test. If the RTE is loaded on one machine and the application software, including the CPE system, is loaded on another, each issues a call for time to their respective system. It is almost impossible to set the two system clocks to within a second of each other. The clocks are accurate to eight millesecond intervals. If the clocks are not reasonably synchronized, the RTE report can show a stimulus sent from the RTE after the SUT has received it. A potential solution would be to have one machine request the time from the other when setting the clock, but the delay of communication from one machine to the other would also affect the time set.

A second, and more significant problem, was the RTE's tendency, during the stress test, to drop one of the logical lines being emulated over the physical line. When it dropped one of the logical lines being emulated, there was no notification to the test director by the RTE software. The loss of a line was not discovered until either the SUT issued a timeout warning message at the operator's console, or the RTE report was examined at the end of the session. Even when the warning message indicated the loss of a line; the line could not be restarted. Several attempts were needed to get valid results for the four terminal emulation. Expanding the software to more than four terminals is likely to increase the occurrence of lost lines.

No valid reason was discovered during the testing to explain the loss of the line. The DLM IV data scope used by the test team did not have a print capability or an extensive memory to store a complete test. By utilizing a different data scope, the reason for the loss of a line should become apparent.

The largest problem discovered during the stress test evaluation was a significant time delay between the transmission of the stimulus by the RTE and its receipt by the SUT. This delay occurred both during the one machine and two machine emulations. For example, the XLOG report (page E-2) shows the transmission of the first stimulus at

12:35:26. The type 10 record from the CPE report (page E-5) shows the stimulus arrived at 12:35:42, a delay of 16 seconds. The second stimulus gives a 14 second delay. All stimuli appear to be delayed between 14 to 16 seconds. Evaluation of the stress test data did not provide the reason for this delay. It is expected that a wait state is occurring in one of the programs.

Summary

While several problems were discovered, only the final one would severely impact the RTE's ability to conduct a valid stress test. The evaluation showed that the RTE could emulate terminals faster than the SUT application software could handle the requests. It also allowed the system test director to simulate the system with both CARGO and PACS terminals in any combination of four terminals. Further refinements of the RTE software, expanding beyond four terminals and eliminating the time delay, will allow the sponsor to stress the application software at will.

VI. Conclusions and Future Studies

Conclusions

This research effort combined the modification of software designed by a civilian contractor with the hardware of the sponsor to determine if the combination would result in an effective tool for software validation and stress testing. The effort was a continuation of concepts in configuration management originally presented by the sponsor in 1982.

The sponsor is determined to find a tool which, when incorporated into his automated release facility, will enable him to improve the reliability of software sent to the operational sites. This study showed that the remote terminal emulator, supplied by the Service Directorate of the Joint Data Systems Support Center, is just the tool they need. It is individually tailored for their type of equipment and the hardware necessary for extensive testing is already in place. The RTE application software package is small enough to reside on the sponsor's hardware with the SUT application software. The cabling connections necessary to utilize the RTE are simple enough to be done without the help of a hardware engineer.

The evaluation of the RTE package, as a software validation tool, showed that the RTE was superior to the sponsor's internal test driver. The RTE software is so easy

to use that the package should be considered as a tool for development testing as well as release testing. While the RTE is deficient in the area of scenario management, it makes up for the deficiency by exercising all the software in the SUT, including the communication handlers.

The evaluation of the RTE package, as a stress testing tool, pointed out that this package allows the sponsor to conduct tests not currently available with his internal test driver. The RTE application software can transmit stimuli faster than the sponsor's SUT application modules can generate the responses, thereby causing a queue of stimuli waiting to enter the SUT application programs. The RTE also allows the system test director to load the SUT with stimuli from more than one of his application systems concurrently. This provides him with an excellent means to test his backup configurations. Elimination of minor problems with the RTE software package will allow the system test director to perform computer performance evaluation studies on the application software before release to the field.

Future Studies

The RTE package was shown to be a quality tool for use by the system test director in releasing software to his operational sites. Just by using the tool in day to day activities of software configuration management, the sponsor will no doubt find many new uses for the RTE package. Possible areas of use for the sponsor include capacity

planning for future sites, hardware acceptance testing for sites with dual machines located in the same area, and benchmark testing for future hardware acquisitions. The Joint Data Systems Support Center briefed the availability of a full screenface emulator at the 1987 Level-6 Terminal User's Conference. The proliferation of this software to other DPS-6 users is bound to open new areas of study which utilize the RTE.

The possible area of study which would constitute a follow-on thesis, however, is not in the use of the RTE package with other DPS-6 machines. The future focus of RTE study, should be directed at using the DPS-6 as the RTE resident machine to emulate and stress other pieces of hardware.

The sponsor, for example, has several configurations which currently could benefit from the use of the RTE. Figure 10 shows a group of current systems which could be driven by the RTE. By loading the RTE software on a DPS-6 which resides before the Datanet Frontend, or on the Datanet, if it is a DPS-6, the system manager could exercise each individual system. Most of these systems are located in the same general area so hardware connections would not be a large problem. By exercising the individual systems, the test director could also exercise the data bases connected to those systems.

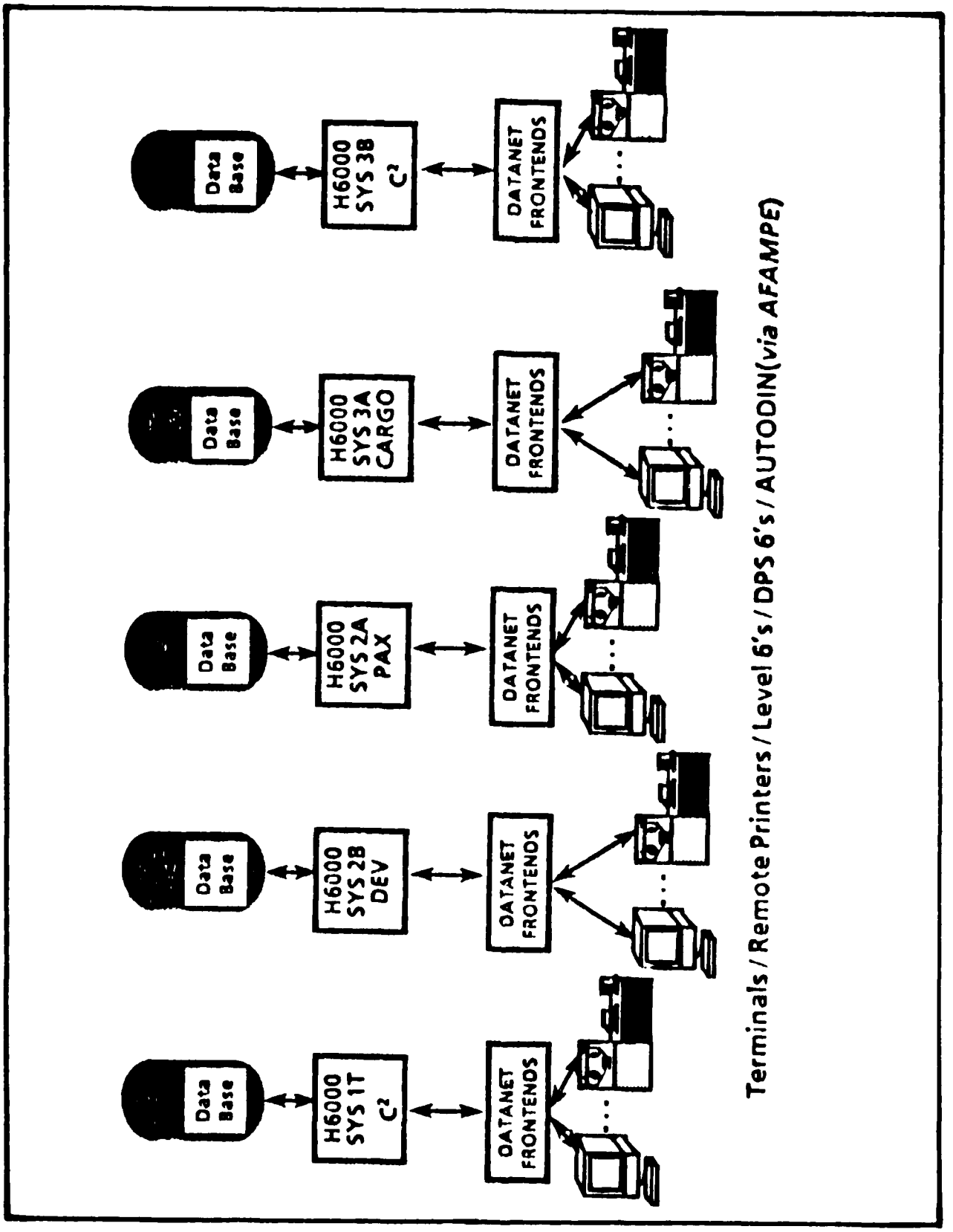


Figure 10. Current System Configuration

Figure 11 shows a future configuration, using a high speed bus to connect computer systems with a general data base. A potentially rewarding study would be the integration of a RTE loaded DPS-6 as one of the 'other hosts' connected to the high speed bus. In this configuration, the RTE could load each of the individual systems, as well as loading the data access processor without use of the individual hosts.

In summary, future studies using the RTE are limited only by the desire of the researcher, the availability of the hardware for research, and the need for the information in respect to the funds made available by the sponsor.

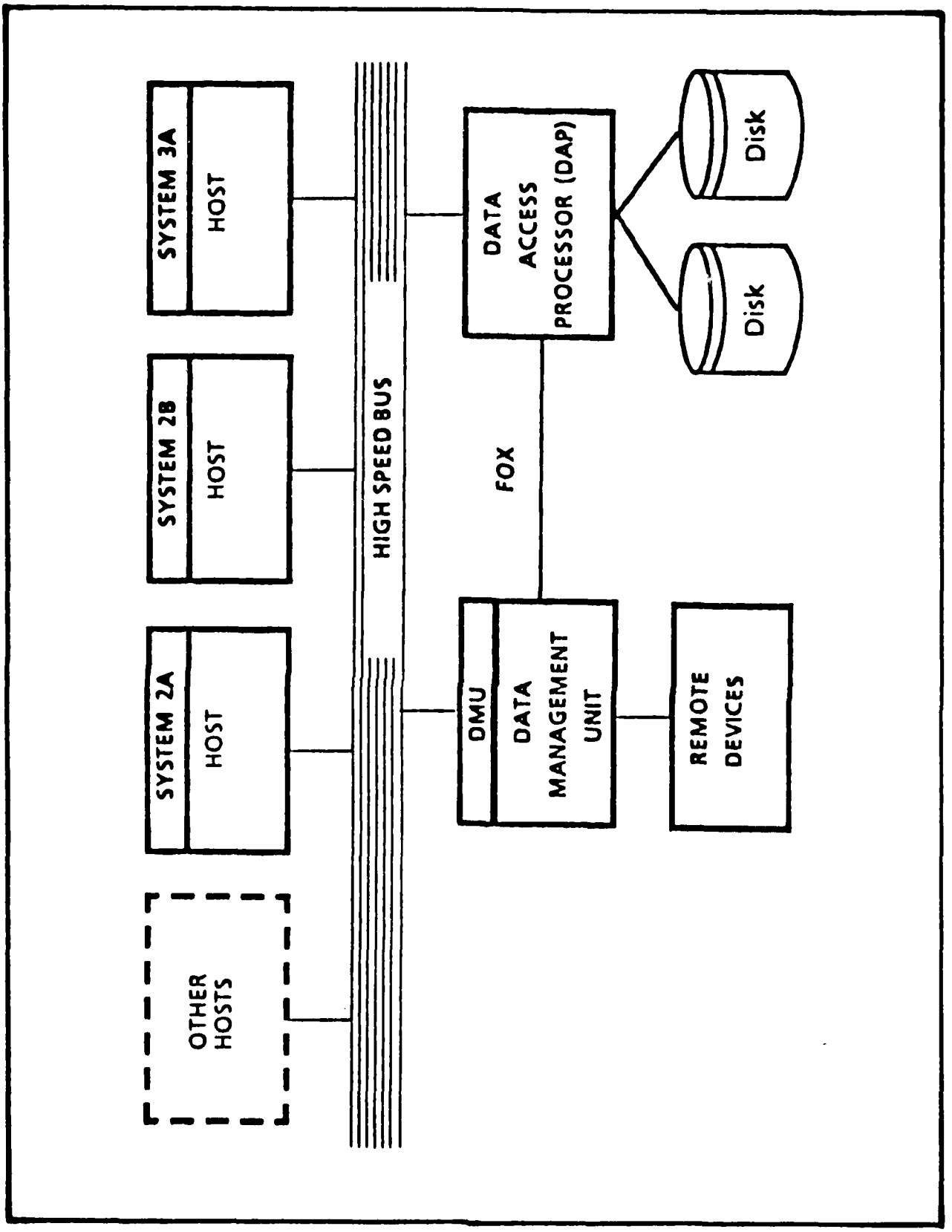
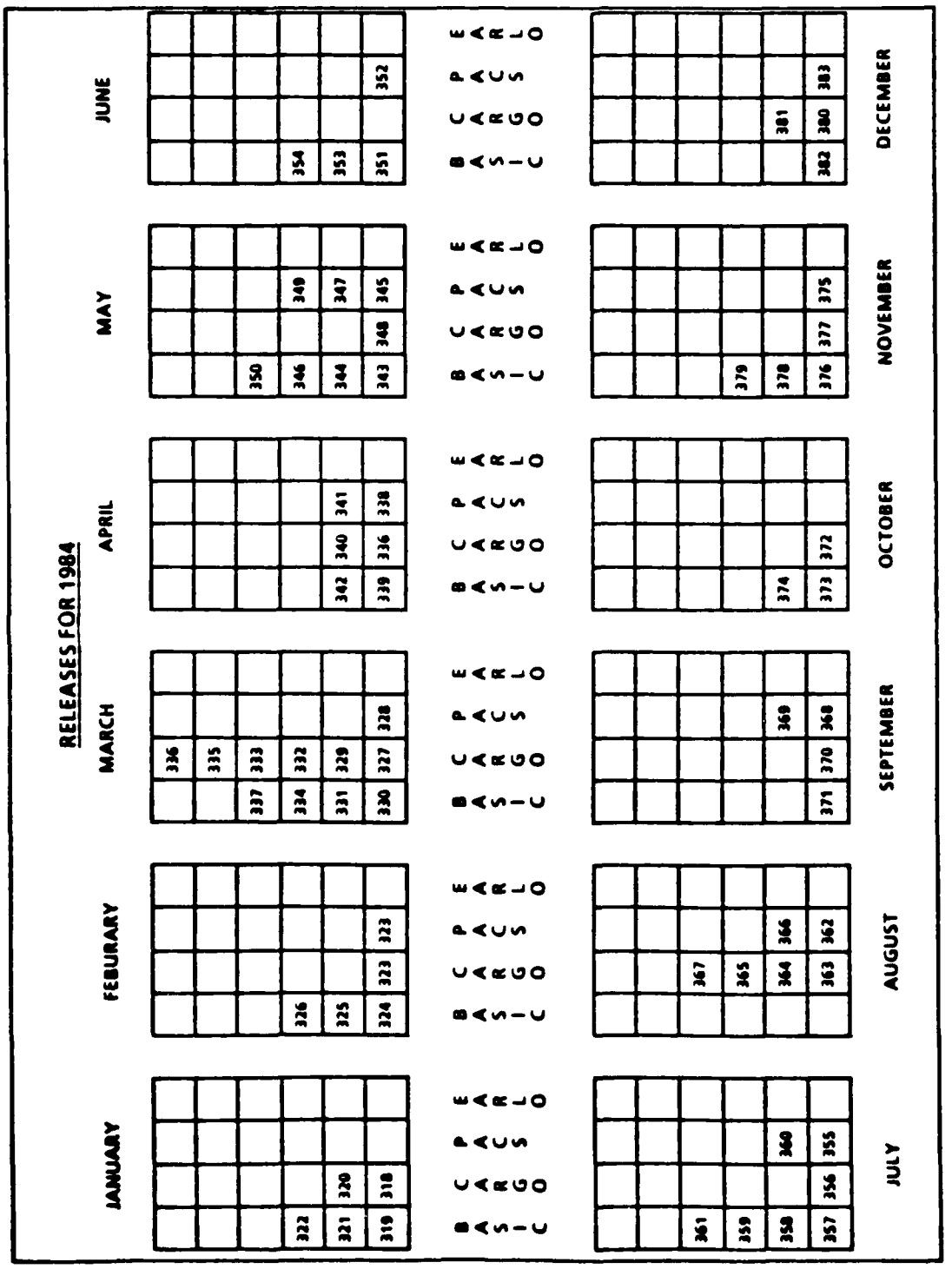
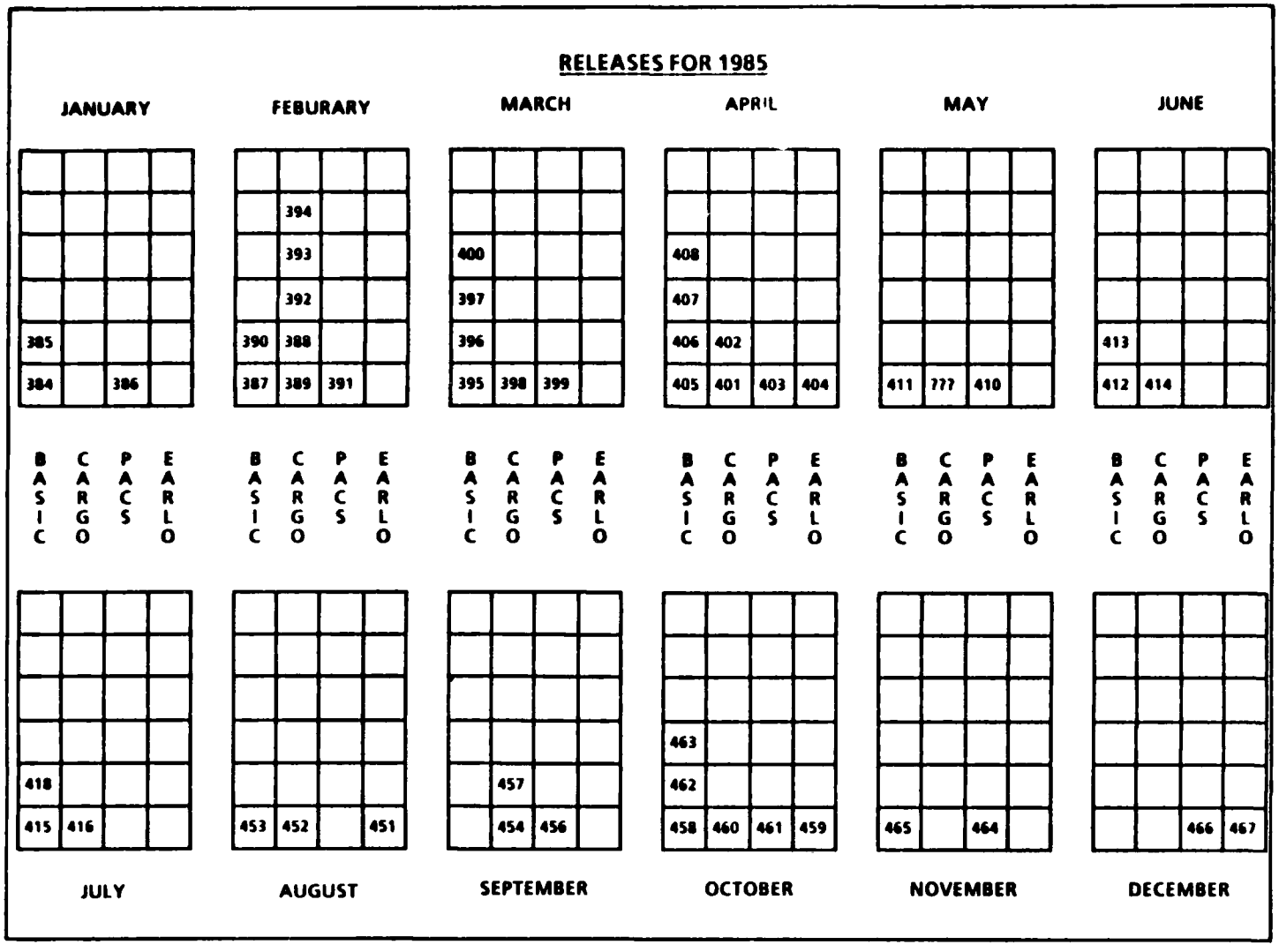
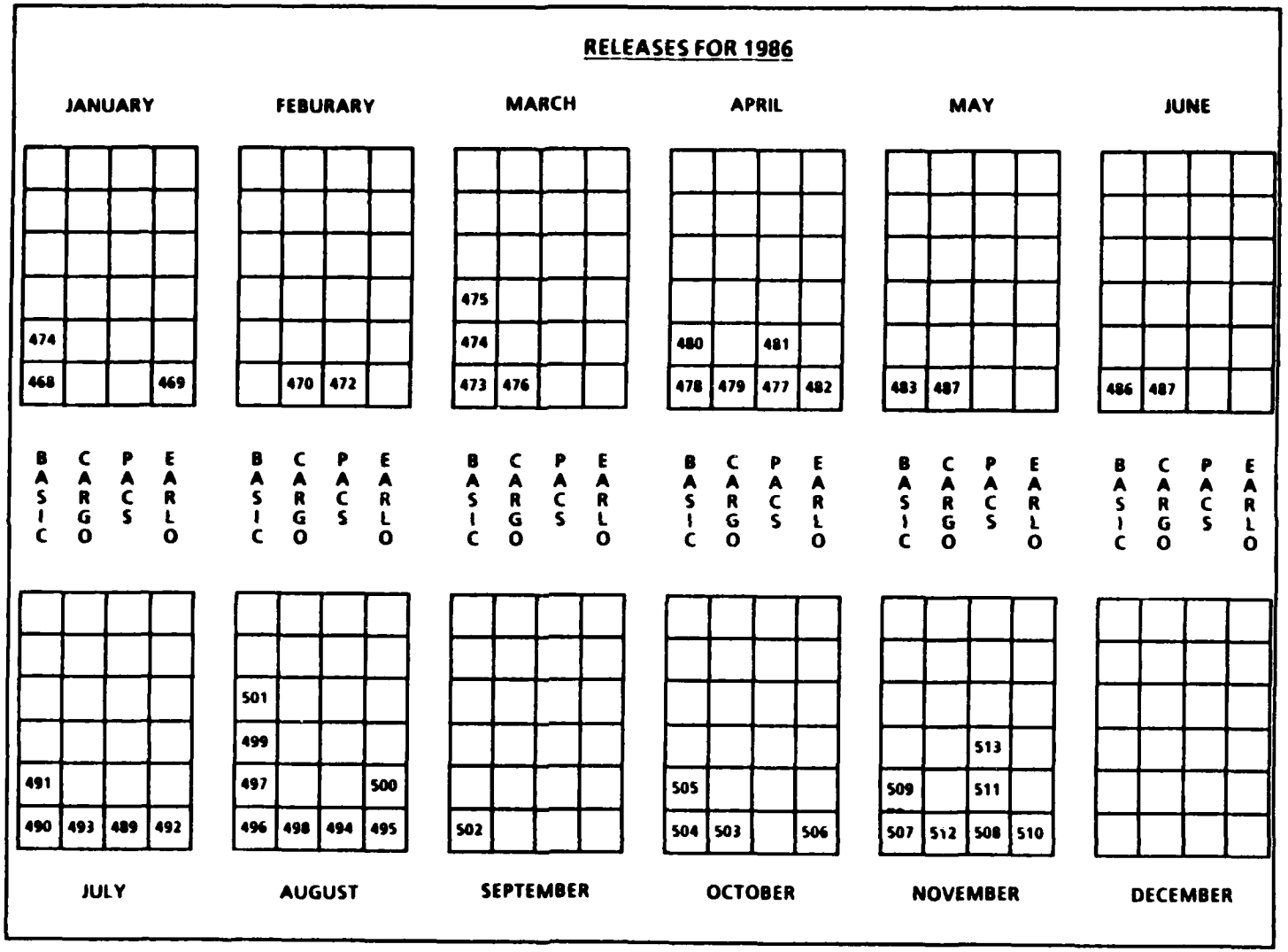


Figure 11. Potential Future System Configuration

Appendix A: Sponsor's Release Data 1984 - 1986



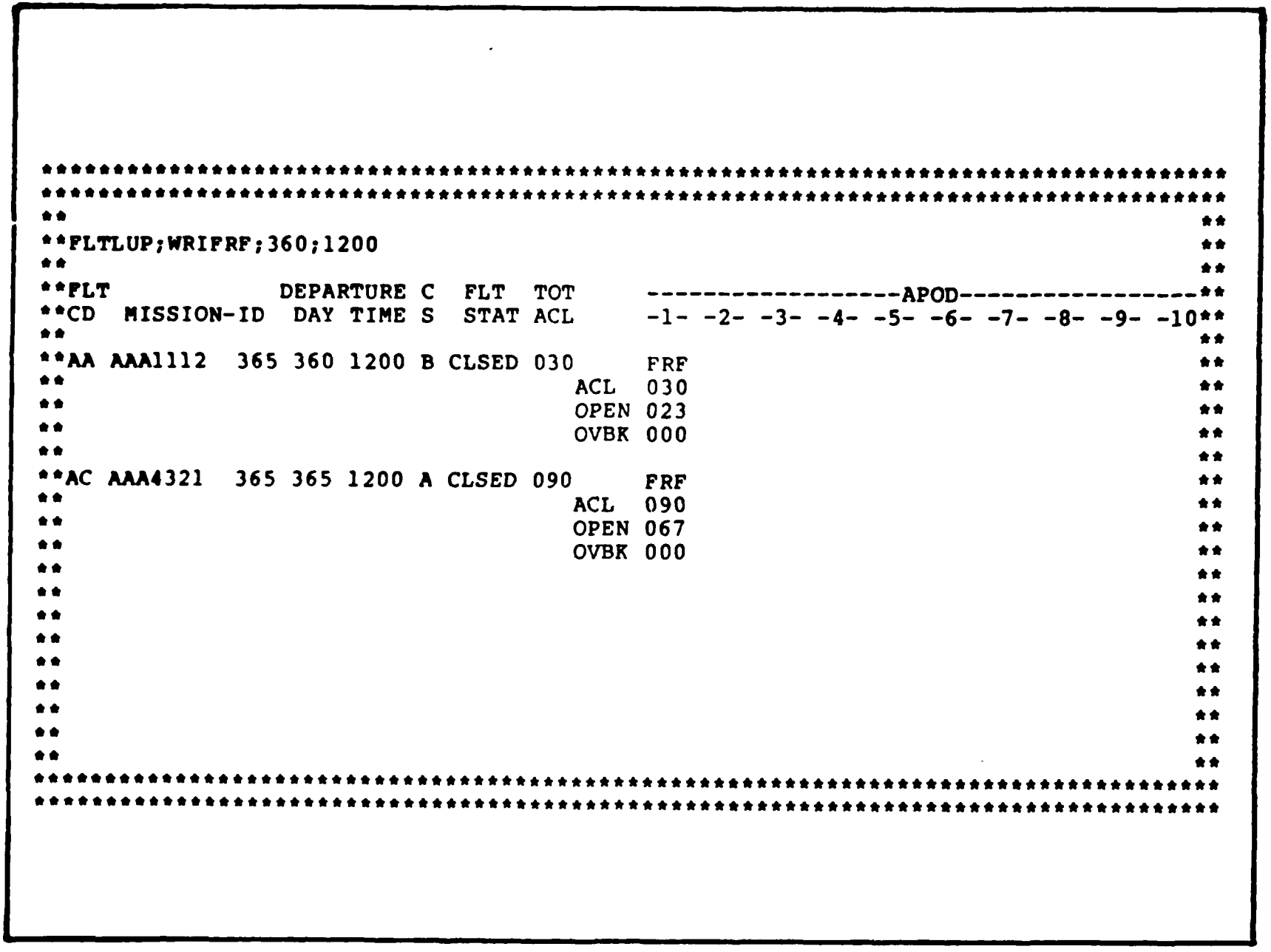

RELEASES FOR 1985
JANUARY
FEBURARY
MARCH
APRIL
MAY
JUNE
385
384
386
394
393
392
390
388
387
389
391
400
397
396
395
398
399
408
407
406
402
405
401
403
404
411
777
410
413
412
414
BASIC
CARGO
PACS
EARLO
418
415
416
453
452
451
457
454
456
463
462
458
460
461
459
465
464
466
467
JULY
AUGUST
SEPTEMBER
OCTOBER
NOVEMBER
DECEMBER

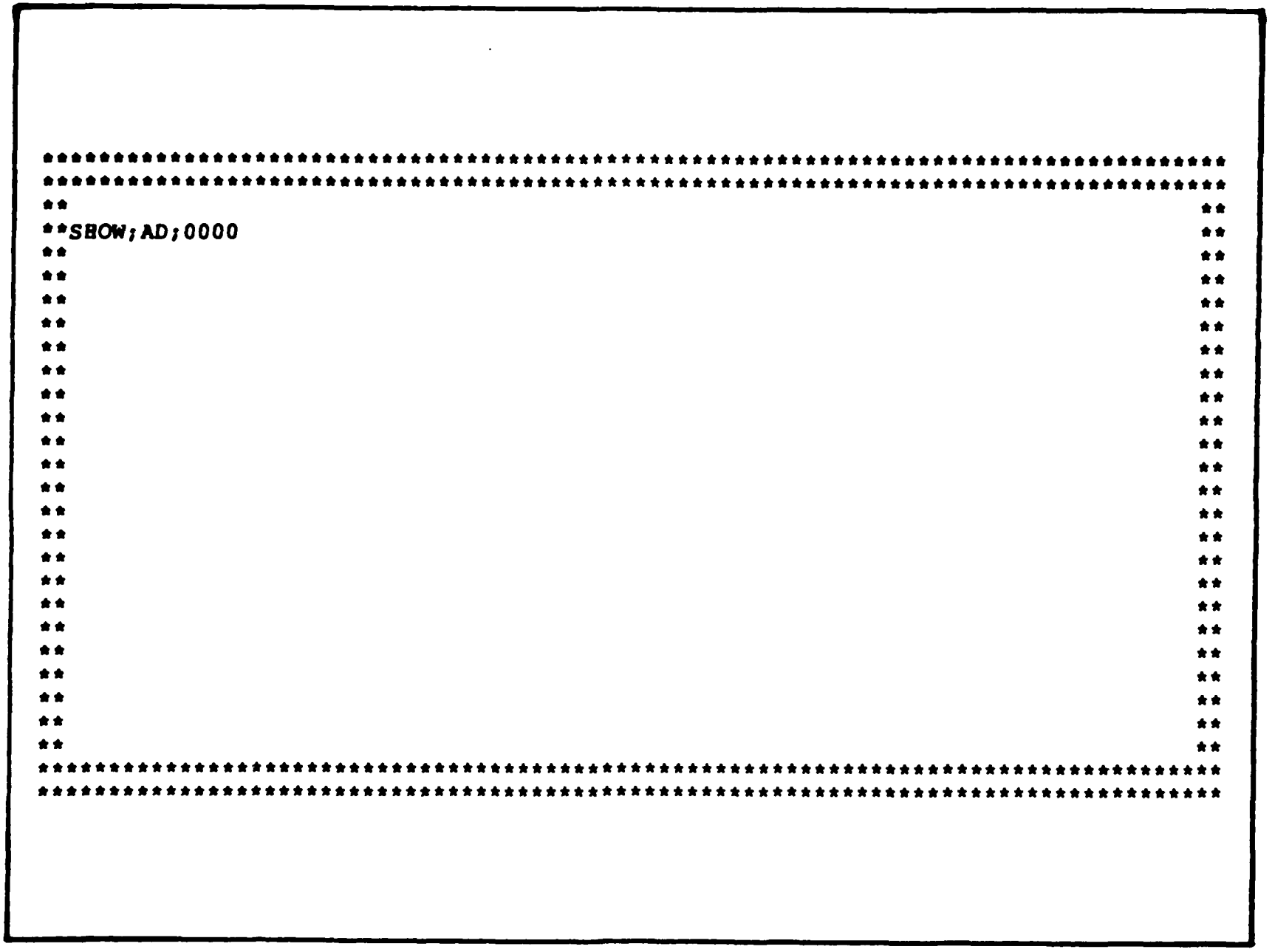

RELEASES FOR 1986
JANUARY
FEBURARY
MARCH
APRIL
MAY
JUNE
474
468
469
470
472
475
474
473
476
480
481
478
479
477
482
483
487
486
487
BASIC
CARGO
PACS
EARLO
491
490
493
489
492
501
499
497
500
496
498
494
495
502
505
504
503
506
513
509
511
507
512
508
510
JULY
AUGUST
SEPTEMBER
OCTOBER
NOVEMBER
DECEMBER

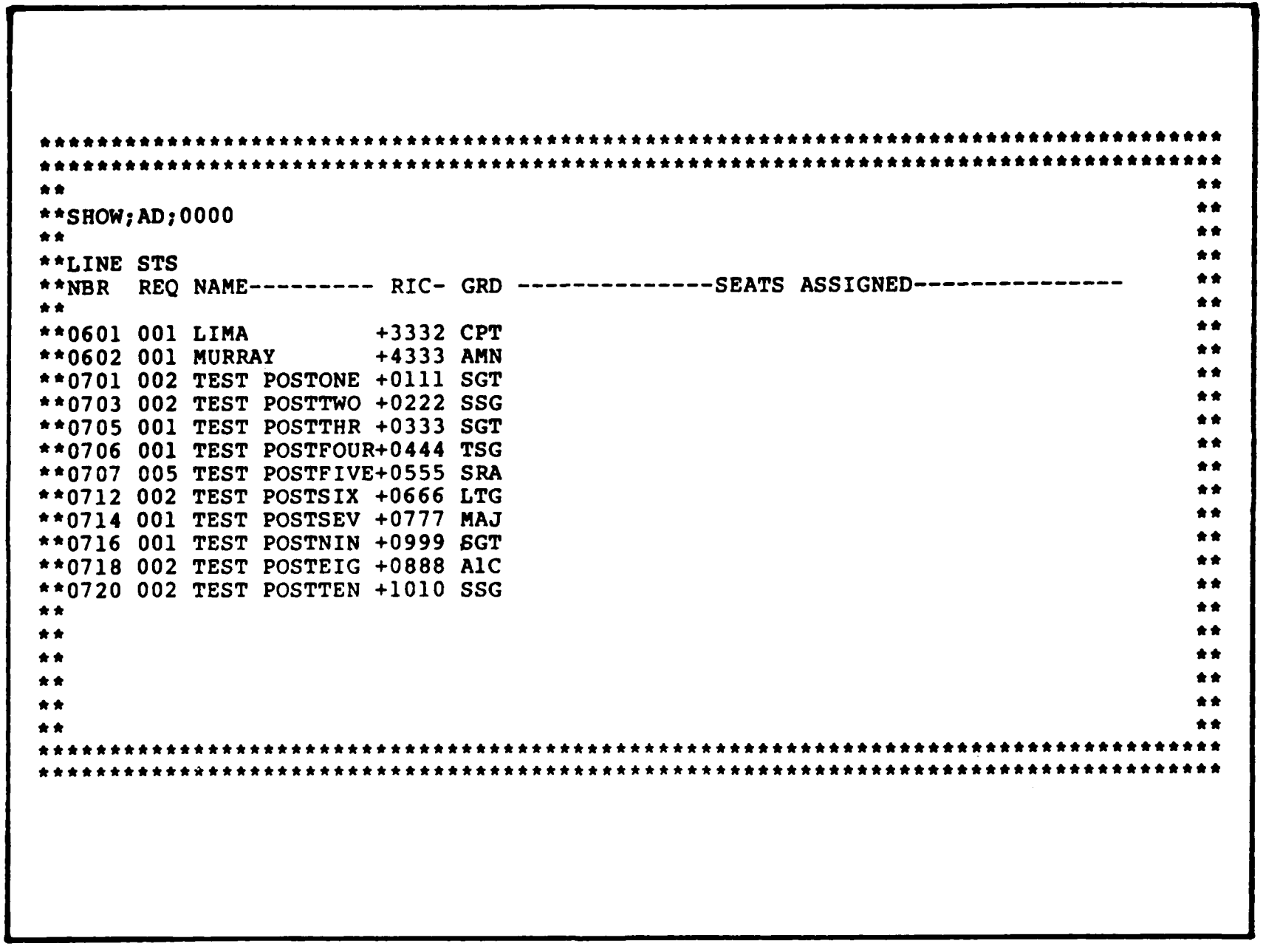
Appendix B: Examples of Inputs and Outputs

This appendix contains two examples of PAX input transacions (B - 2 and B - 4) followed by their output transactions (B - 3 and B - 5).

```
**********************************************************************************************
**********************************************************************************************
**                                                                                          **
**FLTLUP;WRIFRP;360;1200                                                                    **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**                                                                                          **
**********************************************************************************************
**********************************************************************************************
```


FLTLUP;WRIFRF;360;1200
FLT DEPARTURE C FLT TOT APOD
CD MISSION-ID DAY TIME S STAT ACL -1- -2- -3- -4- -5- -6- -7- -8- -9- -10
AA AAA1112 365 360 1200 B CLSED 030 FRF
ACL 030
OPEN 023
OVBK 000
AC AAA4321 365 365 1200 A CLSED 090 FRF
ACL 090
OPEN 067
OVBK 000

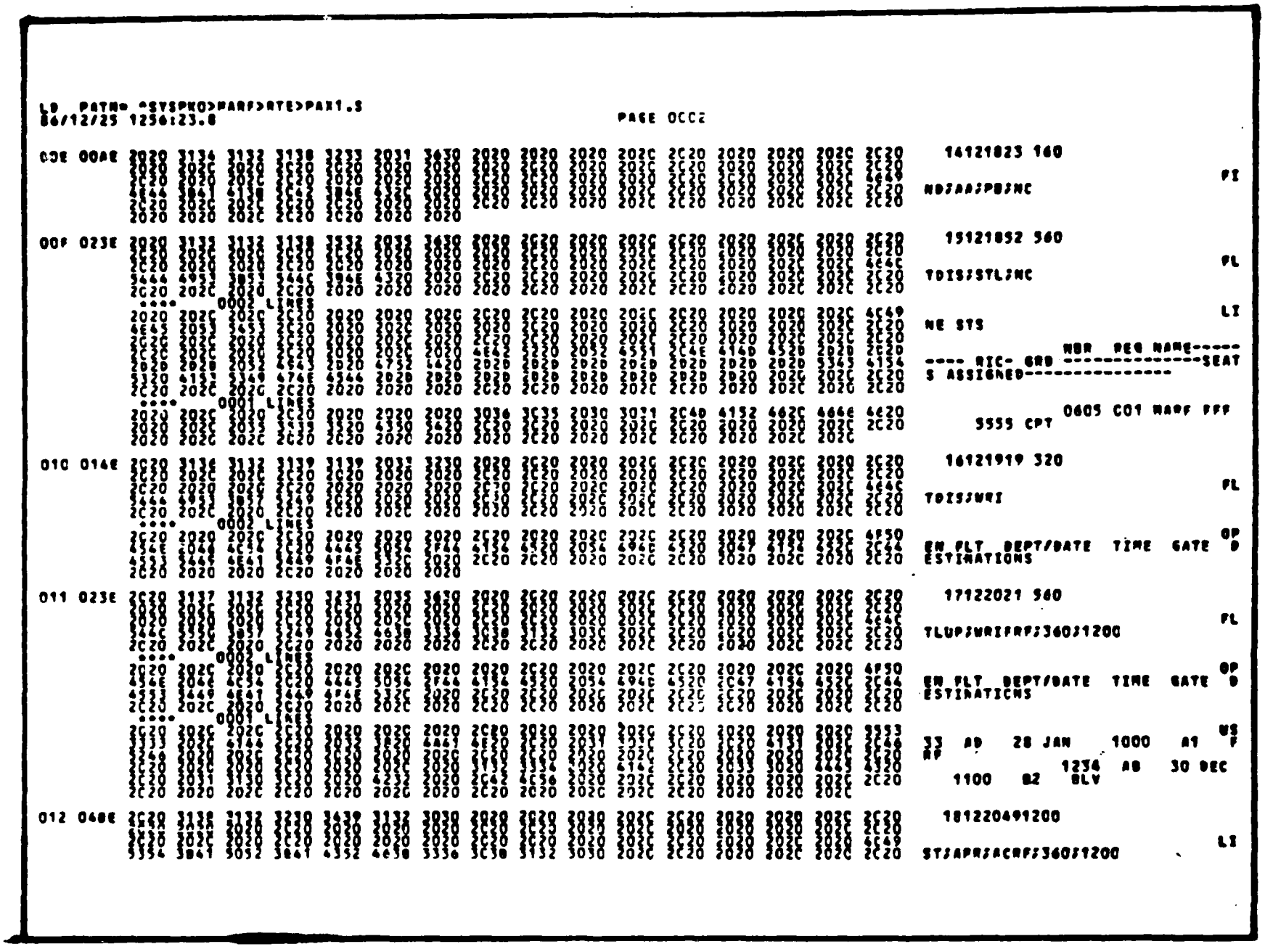

**SHOW;AD;0000


**SHOW;AD;0000
**LINE STS
**NBR REQ NAME--------- RIC- GRD ----------SEATS ASSIGNED----------
**0601 001 LIMA +3332 CPT
**0602 001 MURRAY +4333 AMN
**0701 002 TEST POSTONE +0111 SGT
**0703 002 TEST POSTTWO +0222 SSG
**0705 001 TEST POSTTHR +0333 SGT
**0706 001 TEST POSTFOUR+0444 TSG
**0707 005 TEST POSTFIVE+0555 SRA
**0712 002 TEST POSTSIX +0666 LTG
**0714 001 TEST POSTSEV +0777 MAJ
**0716 001 TEST POSTNIN +0999 SGT
**0718 002 TEST POSTEIG +0888 A1C
**0720 002 TEST POSTTEN +1010 SSG

Appendix C: Printouts of RTE Generated Files

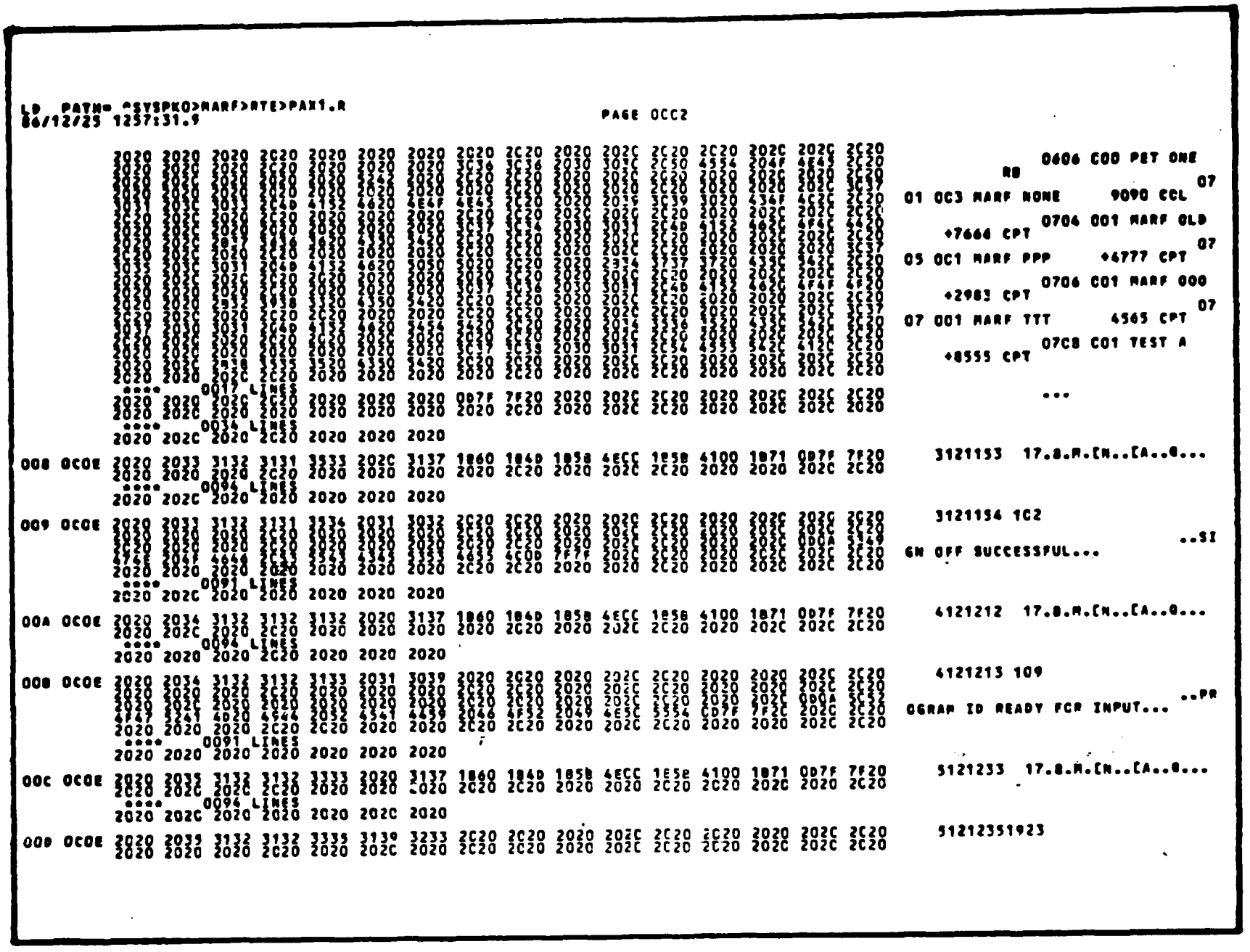
This appendix contains file dumps of a stimulus file (C - 2 and C - 3), file dumps of a response file (C - 4 and C - 5) and a print of the generated script file (C - 6 and C - 7).

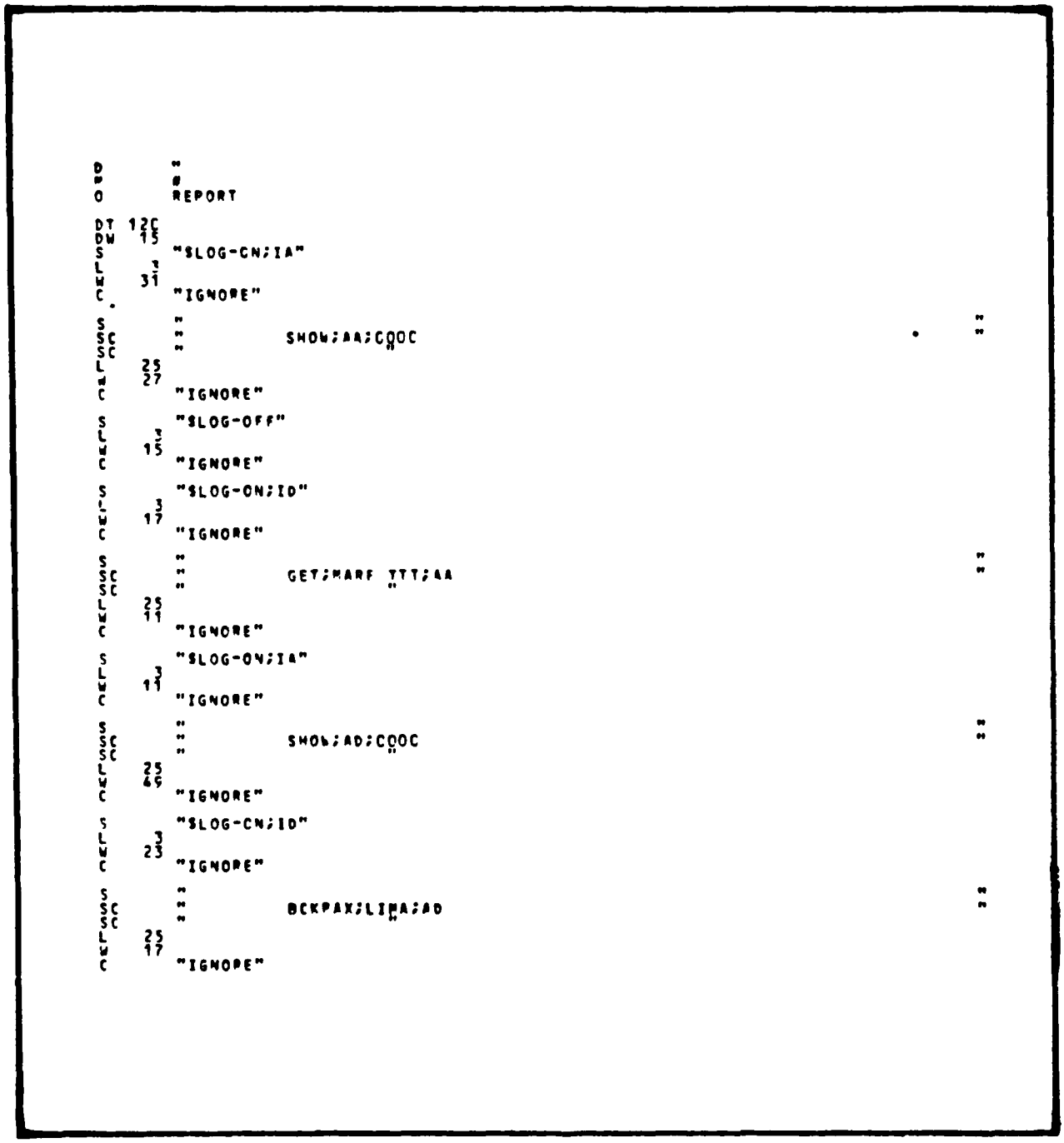
```
LD  PATH= ^SYSPRO>MARF>RTE>PAX1.S
86/12/25 1256:23.8                                          PAGE 0CC1

001 0018 2020 2031 3132 3130 3432 2020 3130 244C 4F47 2D4F 4E3B 4941                    1121042  10SLOG-CN;IA

002 00AE 2C20 2032 3132 3131 3136 2031 3630 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    2121116 160
         2020 2020 202C 2020 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2C20 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 5148                    SM
         2F57 3B41 4138 4C50 5250 202C 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    OW;AA;0000
         2020 202C 202C 2C20 2020 202C 2020 2020 2C20 2020 202C 2C20 202C 2020 202C 2C20
         2020 202C 2020 2020 2020 2020 2020

003 0016 2020 2033 3132 3131 3530 2020 2038 244C 4F47 2D4F 4646                    3121150   8SLOG-OFF

004 0018 2020 2034 3132 3132 3039 2020 3130 244C 4F47 2D4F 4E3B 4944                    41212C9  10SLOG-ON;ID

005 00AE 2020 2033 3132 3132 3130 2031 3630 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    5121230 160
         2C20 2030 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2C20 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 4745                    GE
         3A3B 4B41 5246 2C54 3A54 3B41 4120 2C20 2C20 2020 202C 2C20 2C20 2020 202C 2C20    T;MARF TTT;AA
         2C20 2020 202C 2C20 2020 2020 2020 2C20 2C20 2020 2020 202C 2C20
         2020 202C 202C 2020 2020 2020 2020

006 0018 2020 2036 3132 3132 3436 202C 3130 244C 4F47 2D4F 4E3B 4941                    6121246  10SLOG-CN;IA

007 00AE 2020 2037 3132 3132 3139 2031 3630 2C20 2C20 2020 202C 2C20 2C20 202C 202C 2C20    7121259 160
         2020 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2020 2020 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 5148                    SM
         4F57 3B41 443B 5041 5250 2020 2020 2020 2020 2020 202C 2C20 2C20 202C 202C 2C20    OW;AB;00C0
         2C20 2020 202C 2C20 2020 202C 2020 2020 2020 2C20 2020 202C 2C20 2020 202C 2C20
         2020 202C 2020 2C20 2020 202C 2020

008 0018 2C20 2038 3132 3133 3533 2020 3130 244C 4F47 2D4F 4E3B 4944                    8121353  10SLOG-ON;ID

009 00AE 2C20 2039 3132 3134 3230 2031 3630 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    9121420 160
         2020 202C 2020 2020 202C 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2020 202C 202C 2020 2C20 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 4243                    BC
         4B50 4158 3B4C 494D 413B 4144 2020 2020 2C20 2020 202C 2C20 2020 202C 202C 2C20    KPAX;LIMA;AD
         2C20 202C 2020 2C20 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2020 2020 2020 2020 2020 2020 2020

00A 0018 2020 3130 3132 3134 3431 2020 3130 244C 4F47 2D4F 4E3B 4941                    10121441  10SLOG-ON;IA

00B 00AE 2C20 3131 3132 3135 3132 2031 3630 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    11121512 160
         2C20 2020 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2020 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 4649                    FI
         4E44 3B41 4138 4141 3B4E 4320 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    ND;AA;SA;NC
         2020 2020 2020 2C20 2020 2020 2020 2C20 2C20 2020 2020 2020 2020 2C20
         2C20 202C 2020 2C20 2020 202C 2020

00C 0C5E 2C20 3132 3132 3137 3031 2020 3830 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    121217C1  80
         2020 202C 2020 2C20 2C20 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2020 2020 202C 2C20 2020 202C 2020 2020 2020 202C 202C 2C20 2020 202C 2020

0CB 00AE 2C20 3133 3132 3137 3238 2031 3630 2C20 2C20 2020 2020 2C20 2020 2020 202C 2C20    13121728 160
         2C20 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 2020 202C 2C20
         2C20 2020 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 4243                    BC
         3B55 5341 2020 2C20 2020 202C 2020 2020 2C20 2020 202C 2C20 2020 2045 5252 4F52    ;USA                ERROR
         2049 4E20 4348 4F20 4F52 4445 5220 4941 2020 2020 2C20 2C20 202C 202C 2C20    IN COMMAND FOR IA
         2C20 2020 2020 2020 2020 2020 2020
```

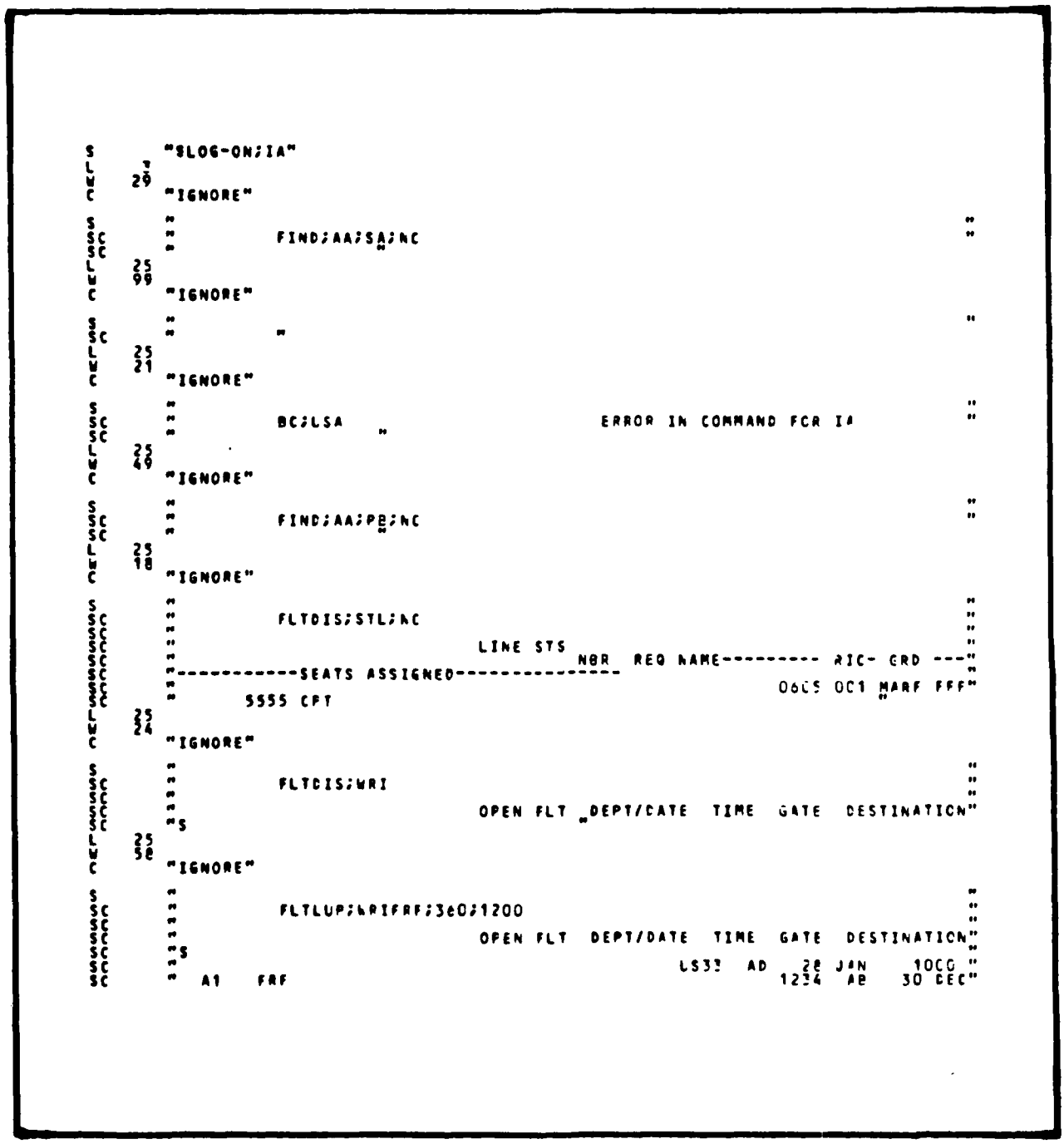


```
LD  PATH= ^SYSPKO>MARF>RTE>PAX1.S
86/12/25 1256:23.8                                                    PAGE 0002

00E 00AE 2020 3134 3132 3138 3233 2031 3630 2020 2020 2020 202C 2C20 2020 2020 202C 2C20    14121823 160
         2020 202C 2020 2C20 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 2020 202C 2C20
         2C20 2020 202C 2C20 2020 2020 2020 2C20 2C20 2020 2020 2C20 2C20 2020 202C 4649                                  FI
         4E44 3B41 4138 3C42 3B4E 432C 2020 2020 2020 2020 202C 2C20 2020 202C 202C 2C20    ND;AA;PB;NC
         2C20 202C 202C 2C20 2C20 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2020 2020 202C 2C20 2C20 2020 2020

00F 023E 2020 3135 3132 3138 3532 2035 3630 2020 2C20 2020 202C 2C20 2020 202C 2020 2C20    15121852 560
         2C20 202C 2020 2C20 2C20 2020 2020 2020 2C20 2020 202C 2C20 2020 2020 2020 2C20
         2C20 2020 2020 2C20 2C20 2020 2020 2020 2C20 2020 202C 2C20 2020 202C 202C 4646                                  FL
         5444 4953 3B53 544C 3B4E 4320 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20    TDIS;STL;NC
         2C20 202C 2020 2C20 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 202C 202C 2C20
          ****     0002 LINES
         2020 202C 202C 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 2020 202C 4C49                                  LI
         4E45 2053 5453 2C20 2020 202C 2020 2020 2C20 2020 2020 2C20 2020 2020 202C 2C20    NE STS
         2C20 202C 2020 2C20 2020 202C 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         2C20 202C 2020 2C20 2020 202C 2020 4E42 5220 2052 4551 2C4E 414D 452D 2D2D 2D2D                  NBR  REQ NAME-----
         2D2D 2D2D 2052 4543 2D20 4752 4420 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 5345 4154    ---- RIC- GRD ---------------SEAT
         5320 4153 5349 474E 4544 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D20 202C 202C 2C20    S ASSIGNED---------------
         2C20 202C 202C 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
          ****     0001 LINES
         2020 202C 2020 2C20 2020 2020 2020 3036 3C35 2030 3031 2C4D 4152 462C 4646 4620                  0605 C01 MARF FFF
         2020 202C 2035 3535 3520 4350 5420 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20        5555 CPT
         2020 202C 202C 2C20 2C20 2020 2020 2020 2C20 2020 202C 2C20 2020 202C 202C

010 014E 2C20 3136 3132 3139 3139 2033 3230 2020 2C20 2020 202C 2C20 2020 202C 2020 2C20    16121919 320
         2C20 202C 202C 2C20 2C20 2020 2020 2C20 2C20 2020 2020 2C20 2C20 202C 202C 2C20
         2C20 2020 202C 2C20 2020 2020 2020 2C20 2C20 202C 202C 2C20 2020 202C 202C 4646                                  FL
         5444 4953 3B57 5249 2C20 2020 2020 2C20 2C20 202C 202C 2C20 2020 202C 202C 2C20    TDIS;WRI
         2C20 202C 2020 2C20 2C20 202C 2020 2020 2C20 2020 202C 2C20 2020 202C 202C 2C20
          ****     0002 LINES
         2C20 2020 202C 2C20 2020 2020 2020 2C20 2020 202C 202C 2C20 2020 2020 202C 4F50                                  OP
         454E 2046 4C54 2C20 4445 5054 2F44 4154 4520 2054 494D 4520 2047 4154 452C 2C44    EN FLT  DEPT/DATE  TIME  GATE  D
         4553 5449 4E41 5449 4F4E 532C 2020 2C20 2C20 2020 202C 2C20 2020 202C 2020 2C20    ESTIMATIONS
         2C20 2020 2020 2C20 2020 2020 2020

011 023E 2C20 3137 3132 3230 3231 2035 3630 2020 2C20 2020 202C 2C20 2020 2020 202C 2C20    17122021 560
         2020 202C 202C 2C20 2C20 2020 2020 2C20 2020 2020 202C 2C20 2020 202C 202C 2C20
         2020 2020 2020 2C20 2020 202C 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 4646                                  FL
         544C 5550 3B57 5249 4452 4638 3336 3C38 3132 3030 202C 2C20 2020 202C 202C 2C20    TLUP;WRIFRF;360;1200
         2C20 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
          ****     0002 LINES
         2C20 202C 2020 2C20 2020 202C 2020 2020 2C20 2020 202C 2C20 2020 202C 2020 4F50                                  OP
         454E 2046 4C54 2C20 4445 5054 2F44 4154 4520 2054 494D 4520 2047 4154 452C 2C44    EN FLT  DEPT/DATE  TIME  GATE  D
         4553 5449 4E41 5449 4F4E 532C 2020 2C20 2C20 2020 202C 2C20 2C20 2020 202C 2C20    ESTIMATIONS
         2C20 202C 2020 2C20 2020 202C 2020 2C20 2C20 2020 202C 2C20 2C20 2020 202C 2C20
          ****     0001 LINES
         2C20 202C 202C 2C20 2020 202C 2020 2C20 2020 2020 202C 2C20 2020 2020 2020 5553                                  US
         3333 202C 4144 2C20 2032 3820 4A41 4E20 2C20 2031 3030 2C20 2C20 4131 202C 2C46    33  AD   28 JAN    1000   A1   F
         5246 2020 202C 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 2020 202C 2C20    RF
         2C20 202C 2020 2C20 2020 202C 2020 3132 3334 3020 4142 2C20 2033 3020 4445 4320                  1234  AB   30 DEC
         2C20 2031 3130 3C20 2020 4232 2020 2C42 4C56 2020 202C 2C20 2020 2020 202C 2C20      1100   B2   BLV
         2C20 2020 202C 2C20 2020 202C 2020 2C20 2020 202C 202C 2C20 2020 2020 202C

012 048E 2C20 3138 3132 3230 3439 3132 3030 2020 2020 2020 202C 2C20 2020 2020 202C 2C20    181220491200
         5450 5050 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2020 2020 2020 202C 2C20
         2C20 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 2020 4C49                                  LI
         5354 3B41 5052 3B41 4352 4638 3336 3C38 3132 3030 202C 2C20 2020 202C 202C 2C20    ST;APR;ACRF;360;1200
```

```
LD  PATH= ^SYSPRO>MARP>RTE>PAR1.R
86/12/23 1257:31.9                                        PAGE 0CC1

001 0C0E 2020 2030 3132 3036 3338 2020 3139 1842 1858 4C18 3855 1858 4218 5864 1872 087F    0120658  19.B.CL.CU.CB.CD.R..
         7F30 2020 2020 2C20 2C20 2020 2020 2020 2020 2020 2020 202C 2C20 2020 202C 202C 2C20
         2020 2020 2020 2C20 2C20 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 2020 202C 2C20
         ****  0001 LINES
         2020 2020 202C 2020 2020 2020 2020

002 0C0E 2020 2030 3132 3C34 3338 2020 3137 1840 1858 1858 4E5C 1858 4100 1871 087F 7F20    0120658  17.8.M.CN..CA..0...
         2020 202C 2020 2C20 2020 2020 2020 2C20 2C20 2020 2020 2C20 2020 202C 202C 2C20
         ****  0001 LINES
         2C20 2C20 2020 2020 2020 2020 2020

003 0C0E 2020 2030 3132 3C34 3339 2020 3239 2032 4141 4459 2044 4F52 2043 4F4E 3432 4F46    0120659  29 READY FOR CONTROL
         2049 4E50 3554 2020 2020 4F50 2020 2020 2020 2020 202C 2C20 2020 202C 202C 2C20    INPUT ....
         2020 2020 2020 2C20 2020 2020 2020 2C20 2C20 2020 202C 2C20 2020 202C 202C 2C20
         ****  0001 LINES
         2020 2020 2020 2020 2020 2020 2020

004 0C0E 2020 2031 3132 3130 3434 2020 3137 1840 1858 1858 4E5C 1656 4100 1871 087F 7F20    1121044  17.8.M.CN..CA..0...
         2020 202C 2020 2C20 2C20 2020 2020 2C20 2C20 202C 252C 2C20 2C20 202C 2020 2C20
         ****  0001 LINES
         2020 2020 202C 2020 2020 2020 2020

005 0C0E 2020 2031 3132 3130 3435 2031 3039 2C20 2C20 2020 202C 2C20 2020 202C 2020 2C20    1121045 1C9
         2020 2020 2020 2C20 2020 2020 2020 2C20 2C20 2020 2020 2C20 2C20 202C 202C 2C20
         2020 2020 2020 2C20 2020 2020 2020 2C20 2C20 2C20 202C 2C20 2C20 202C 0D0A 5748    ..PR
         4F47 5241 4D20 4941 2052 4541 4459 2046 4F52 2049 4E50 5554 C0FF 7F2C 202C 2C20    OGRAP IA READY FOR INPUT...
         2020 2020 2020 2C20 2020 2020 2020 2020 2020 2020 202C 2C20 2020 2020 202C 2C20
         ****  0001 LINES
         2020 2020 2020 2020 2020 2020 2020

006 0C0E 2020 2032 3132 3131 3231 2020 3137 1840 1858 1858 4E5C 1858 4100 1871 C87E 7F20    2121121  17.8.M.CN..CA..0...
         2020 2020 2020 2020 2020 2020 2020 2C20 2C20 2020 2020 2C20 2020 202C 202C 2C20
         ****  0001 LINES
         2020 202C 2020 2020 2020 2020 2020

007 0C0E 2020 2032 3132 3131 3233 3139 3231 2020 2C20 2020 202C 2C20 2023 2020 2020 2C20    21211231923
         2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 202C 2C20 2C20 2020 202C 2C20
         2020 2020 2020 2020 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 2020 202C 0D0A 5748    ..SN
         4F57 3841 4138 2C20 2020 2020 2020 2020 4C20 2020 202C 2C20 2020 202C 202C 2C20    OWJAAJOOCO
         2020 2020 2020 2C20 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 202C 202C 2C20
         ****  0002 LINES
         2020 2020 2020 2020 2020 2020 2020 2C20 2020 2020 202C 2C20 2C20 2020 202C 4C49    LI
         4E45 2053 5453 2020 2020 2020 2020 2020 2C20 202C 2020 2C20 2020 202C 202C 2C20    NE STS
         2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2C20 2020 202C 202C 2C20
         2020 2020 2020 2020 2020 2020 2020 4842 5220 2052 4551 2C4E 4146 452D 2D2D 2D2D    NOR  REQ NAME-----
         2D2D 2D2D 2052 4943 2D20 4752 4420 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D20 5345 4154    ---- RIC- GRD ----------SEAT
         5320 4153 5349 4746 4544 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 2D2D 202C 202C 2C20 2C20    S ASSIGNED----------
         2020 2020 2C20 2C20 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 202C 202C 2C20
         ****  0001 LINES
         2020 202C 2020 2C20 2020 2020 2020 3036 3C31 2030 3031 2C40 4152 4620 3457 4F20    0601 001 MARF TWO
         2C20 2020 2B33 3434 3420 4350 5420 2020 2020 2020 2020 2C20 2020 2020 202C 2C20      +3444 CPT
         2020 2020 2020 2C20 2C20 2020 2020 2020 2020 2020 2020 202C 2C20 2020 202C 2C20    06
         5020 5052 5020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2C20    02 001 MARF THREE   +4333 ANN
         2020 2020 2020 2C20 2020 2020 2020 2020 2C20 2020 202C 2C20 2020 2020 202C 2C20    0603 001 MARF FOUR
         2020 202C 2020 2C20 2020 2020 2020 3C36 3033 2030 3031 2C40 4152 462C 464F 5552      +2222 COL
         5050 5052 2B32 3232 3220 434F 4C20 2020 2020 2020 2020 2C20 2020 202C 202C 2C20    06
         5035 2030 3031 2049 4152 4620 4646 4620 2020 2020 2035 3535 3520 435C 5420 2C20    05 C01 MARF FFF      5555 CPT
```


LD PATH= ^SYSPKO>MARF>RTE>PAX1.R
86/12/25 1257:31.9
PAGE 0CC2
0606 C00 PET ONE
RB
07
01 0C3 MARF NONE 9090 CCL
0704 001 MARF OLD
+7664 CPT
07
05 0C1 MARF PPP +4777 CPT
0706 C01 MARF 000
+2983 CPT
07
07 001 MARF TTT 4565 CPT
07C8 C01 TEST A
+8555 CPT
0017 LINES
...
0034 LINES
008 0C0E
3121153 17.8.M.[N..[A..0...
0094 LINES
009 0C0E
3121154 1C2
..SI
GN OFF SUCCESSFUL...
0091 LINES
00A 0C0E
4121212 17.8.M.[N..[A..0...
0094 LINES
00B 0C0E
4121213 109
..PR
OGRAM ID READY FOR INPUT...
0091 LINES
00C 0C0E
5121233 17.8.M.[N..[A..0...
0096 LINES
00D 0C0E
51212351923


REPORT
"SLOG-ON;IA"
"IGNORE"
"IGNORE"
"SLOG-OFF"
"IGNORE"
"SLOG-ON;ID"
"IGNORE"
"IGNORE"
"SLOG-ON;IA"
"IGNORE"
"IGNORE"
"SLOG-ON;ID"
"IGNORE"
"IGNOPE"


"SLOG-ON;IA"
"IGNORE"
FIND;AA;SA;NC
"IGNORE"
"IGNORE"
BC;LSA
ERROR IN COMMAND FOR IA
"IGNORE"
FIND;AA;PB;NC
"IGNORE"
FLTDIS;STL;NC
LINE STS
NBR REQ NAME--------- RIC- GRD ---
----------SEATS ASSIGNED----------
0605 0C1 MARF FFF
5555 CPT
"IGNORE"
FLTDIS;WRI
OPEN FLT DEPT/DATE TIME GATE DESTINATION
"IGNORE"
FLTLUP;ARIFRF;3E0;1200
OPEN FLT DEPT/DATE TIME GATE DESTINATION
LS33 AD 28 JAN 1000
A1 FRF
1234 AB 30 DEC

Appendix D: Screenfaces for Response Verificaion

This appendix contains examples of screenfaces generated by the response verification command of the Autogen module.

```
                                                        DW=  15
TRANSACTION:     1
LABEL: NONE
STIMULUS TEXT:
$LOG-ON;IA
LENGTH OF RESPONSE:    2
WAIT TIME:    31
CONTINGENCY ACTION: IGNORE
COMMAND:
```

```
   1: _`_M_[N__[A__q___

   2:                    __PROGRAM IA READY FOR INPUT___
IS DESIRED STARTING LINE ON THIS PAGE? (Y/N)
ENTER STARTING LINE NUMBER:
```

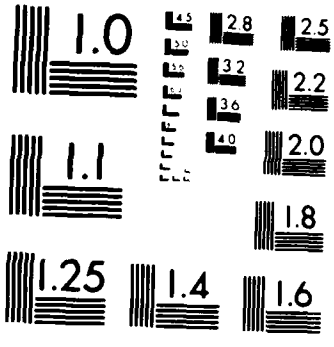
```
!...+....1....+....2....+....3....+....4....+....5....+....6...
_`_M_[N__[A__q___
START OF RESPONSE TEXT:
```

```
   1: _`_M_[N__[A__q___

   2:                    __PROGRAM IA READY FOR INPUT___
IS DESIRED ENDING   LINE ON THIS PAGE? (Y/N)
ENTER ENDING LINE NUMBER:
```

AD-A188 833 MODIFICATION IMPLEMENTATION AND EVALUATION OF A REMOTE TERMINAL EMULATOR A.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C J RIESBERG

2/2

UNCLASSIFIED DEC 87 AFIT/GCS/MA/87D-5 F/G 12/5 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

```
1...+....1....+....2....+....3....+....4....+....5....+....6...
_`_M_[N__[A__q___
END OF RESPONSE TEXT:
```

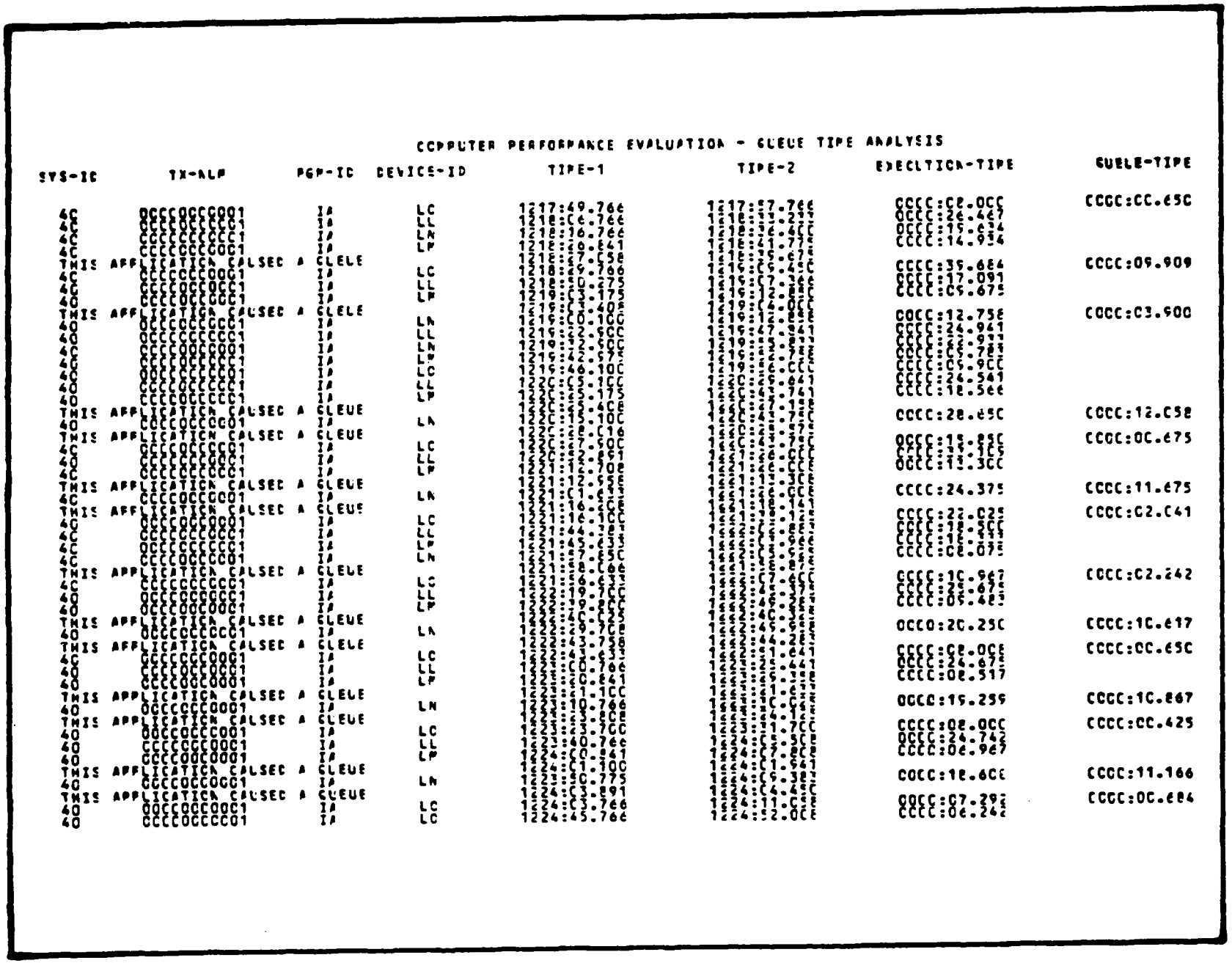
```
                                                        DW=  15
TRANSACTION:     1
LABEL: NONE
STIMULUS TEXT:
$LOG-ON;IA
LENGTH OF RESPONSE:    2
RESPONSE TEXT:
_`_M_[N__[A__q___
WAIT TIME:    31
CONTINGENCY ACTION: IGNORE
COMMAND:
```

Appendix E: Reports Generated During Stress Test

This appendix contains examples of reports generated by the RTE report generator (E - 2 and E - 3) and reports from the sponsor's computer performance evaluation systeem (E - 4 through E - 7).

```
*************   REMCTE TERMINAL EMULATOR -LOG EXPANSION   *************
CODE LINE SCRIPT          TRANSACTICN DESCRIPTION                    TIME
-----------------------------------------------------------------------

 T       1 -   1    TRANSMISSION OF STIMULUS    1                   12:35:26
 R       1 -   1    RESPCNSE TO STIMULUS  1 RECEIVED                12:35:43
                   --  NCT CCMPARED  --
 V       1 -   1    RESPONSE TO STIMULUS   1 VERIFIED               12:35:43
 T       1 -   1    TRANSMISSION OF STIMULUS    2                   12:35:44
 R       1 -   1    RESPCNSE TC STIMULUS  2 RECEIVED                12:36: 2
                   --  NOT CCMPARED  --
 V       1 -   1    RESPCNSE TO STIMULUS   2 VERIFIED               12:36: 2
 T       1 -   1    TRANSMISSION OF STIMULUS    3                   12:36: 2
 R       1 -   1    RESPCNSE TO STIMULUS  3 RECEIVED                12:36:19
                   --  NCT CCMPARED  --
 V       1 -   1    RESPCNSE TO STIMULUS   3 VERIFIED               12:36:19
 T       1 -   1    TRANSMISSION CF STIMULUS    4                   12:36:19
 R       1 -   1    RESPCNSE TO STIMULUS  4 RECEIVED                12:36:35
                   --  NCT CCMPARED  --
 V       1 -   1    RESPCNSE TO STIMULUS   4 VERIFIED               12:36:35
 T       1 -   1    TRANSMISSION OF STIMULUS    5                   12:36:36
 R       1 -   1    RESPCNSE TO STIMULUS  5 RECEIVED                12:36:53
                   --  NCT COMPARED  --
 V       1 -   1    RESPCNSE TO STIMULUS   5 VERIFIEC               12:36:53
 T       1 -   1    TRANSMISSION OF STIMULUS    6                   12:36:53
 R       1 -   1    RESPONSE TO STIMULUS  6 RECEIVED                12:37:1C
```

```
                    --  NOT COMPARED  --
V       1 -  1   RESPONSE TO STIMULUS   6 VERIFIED          12:37:10
T       1 -  1   TRANSMISSION OF STIMULUS    7              12:37:11
R       1 -  1   RESPONSE TO STIMULUS  7 RECEIVED           12:37:29
                    --  NOT COMPARED  --
V       1 -  1   RESPONSE TO STIMULUS   7 VERIFIED          12:37:29
T       1 -  1   TRANSMISSION OF STIMULUS    8              12:37:30
R       1 -  1   RESPONSE TO STIMULUS  8 RECEIVED           12:37:46
                    --  NOT COMPARED  --
V       1 -  1   RESPONSE TO STIMULUS   8 VERIFIED          12:37:46
T       1 -  1   TRANSMISSION OF STIMULUS    9              12:37:47
R       1 -  1   RESPONSE TO STIMULUS  9 RECEIVED           12:38: 4
                    --  NOT COMPARED  --
V       1 -  1   RESPONSE TO STIMULUS   9 VERIFIED          12:38: 4
T       1 -  1   TRANSMISSION OF STIMULUS   10              12:38: 4
R       1 -  1   RESPONSE TO STIMULUS 10 RECEIVED          12:38:20
                    --  NOT COMPARED  --
V       1 -  1   RESPONSE TO STIMULUS  10 VERIFIED          12:38:20
T       1 -  1   TRANSMISSION OF STIMULUS   11              12:38:21
R       1 -  1   RESPONSE TO STIMULUS 11 RECEIVED          12:38:44
                    --  NOT COMPARED  --
V       1 -  1   RESPONSE TO STIMULUS  11 VERIFIED          12:38:44
T       1 -  1   TRANSMISSION OF STIMULUS   12              12:38:45
R       1 -  1   RESPONSE TO STIMULUS 12 RECEIVED          12:39: 2
```



COMPUTER PERFORMANCE EVALUATION - QUEUE TIME ANALYSIS

SYS-ID	TX-ALP	PGP-ID	DEVICE-ID	TIME-1	TIME-2	EXECUTION-TIME	QUEUE-TIME
40	00CCC0CCC001	IA	LC	1217:49.766	1217:57.766	0000:08.000	0000:00.650
40	00CCC0CCC001	IA	LC	1218:06.766	1218:09.222	0000:02.456	
40	00CCC0CCC001	IA	LN	1218:16.766	1218:18.421	0000:01.654	
40	00CCC0CCC001	IA	LP	1218:26.641	1218:28.777	0000:14.934	
THIS	APPLICATION CALSEC A	GUEUE		1218:36.575	1218:39.175		
40	00CCC0CCC001	IA	LC	1218:46.766	1218:49.422	0000:35.664	0000:09.909
40	00CCC0CCC001	IA	LP	1219:06.775	1219:08.866	0000:17.091	
40	00CCC0CCC001	IA	LP	1219:26.775	1219:28.866	0000:09.675	
THIS	APPLICATION CALSEC A	GUEUE		1219:36.450	1219:39.202		
40	00CCC0CCC001	IA	LN	1219:46.766	1219:49.422	0000:12.756	0000:03.900
40	00CCC0CCC001	IA	LC	1219:56.766	1219:59.422	0000:24.941	
40	00CCC0CCC001	IA	LN	1220:06.766	1220:09.422	0000:09.991	
40	00CCC0CCC001	IA	LC	1220:16.766	1220:19.422	0000:09.781	
40	00CCC0CCC001	IA	LP	1220:26.641	1220:29.175	0000:05.900	
40	00CCC0CCC001	IA	LP	1220:36.575	1220:39.175	0000:24.541	
THIS	APPLICATION CALSEC A	GUEUE		1220:46.450	1220:49.202	0000:18.566	
40	00CCC0CCC001	IA	LN	1220:56.766	1220:59.422	0000:28.650	0000:12.058
THIS	APPLICATION CALSEC A	GUEUE		1221:06.766	1221:09.422	0000:15.850	0000:00.675
40	00CCC0CCC001	IA	LC	1221:16.766	1221:19.422	0000:14.765	
40	00CCC0CCC001	IA	LP	1221:26.641	1221:29.175	0000:11.300	
THIS	APPLICATION CALSEC A	GUEUE		1221:36.575	1221:39.175		
40	00CCC0CCC001	IA	LN	1221:46.766	1221:49.422	0000:24.375	0000:11.675
THIS	APPLICATION CALSEC A	GUEUE		1221:56.766	1221:59.422	0000:22.025	0000:02.041
40	00CCC0CCC001	IA	LC	1222:06.766	1222:09.422	0000:18.500	
40	00CCC0CCC001	IA	LP	1222:16.766	1222:19.422	0000:15.333	
40	00CCC0CCC001	IA	LN	1222:26.641	1222:29.175	0000:08.075	
THIS	APPLICATION CALSEC A	GUEUE		1222:36.575	1222:39.175		
40	00CCC0CCC001	IA	LC	1222:46.766	1222:49.422	0000:10.967	0000:02.242
40	00CCC0CCC001	IA	LP	1222:56.766	1222:59.422	0000:24.671	
40	00CCC0CCC001	IA	LP	1223:06.766	1223:09.422	0000:08.483	
THIS	APPLICATION CALSEC A	GUEUE		1223:16.766	1223:19.422		
40	00CCC0CCC001	IA	LN	1223:26.641	1223:29.175	0000:20.250	0000:10.617
THIS	APPLICATION CALSEC A	GUEUE		1223:36.575	1223:39.175		
40	00CCC0CCC001	IA	LC	1223:46.766	1223:49.422	0000:08.000	0000:00.650
40	00CCC0CCC001	IA	LP	1223:56.766	1223:59.422	0000:24.671	
40	00CCC0CCC001	IA	LP	1224:06.766	1224:09.422	0000:08.517	
THIS	APPLICATION CALSEC A	GUEUE		1224:16.766	1224:19.422		
40	00CCC0CCC001	IA	LN	1224:26.641	1224:29.175	0000:19.259	0000:10.867
THIS	APPLICATION CALSEC A	GUEUE		1224:36.575	1224:39.175		
40	00CCC0CCC001	IA	LC	1224:46.766	1224:49.422	0000:08.000	0000:00.425
40	00CCC0CCC001	IA	LP	1224:56.766	1224:59.422	0000:24.743	
40	00CCC0CCC001	IA	LP	1225:06.766	1225:09.422	0000:08.967	
THIS	APPLICATION CALSEC A	GUEUE		1225:16.766	1225:19.422		
40	00CCC0CCC001	IA	LN	1225:26.641	1225:29.175	0000:18.600	0000:11.166
THIS	APPLICATION CALSEC A	GUEUE		1225:36.575	1225:39.175		
40	00CCC0CCC001	IA	LC	1225:46.766	1225:49.422	0000:07.291	0000:00.684
40	00CCC0CCC001	IA	LC	1225:56.766	1225:59.422	0000:08.242	

COMPUTER PERFORMANCE EVALUATION JOURNALIZATION REPORT
HQ MAC/ADCI

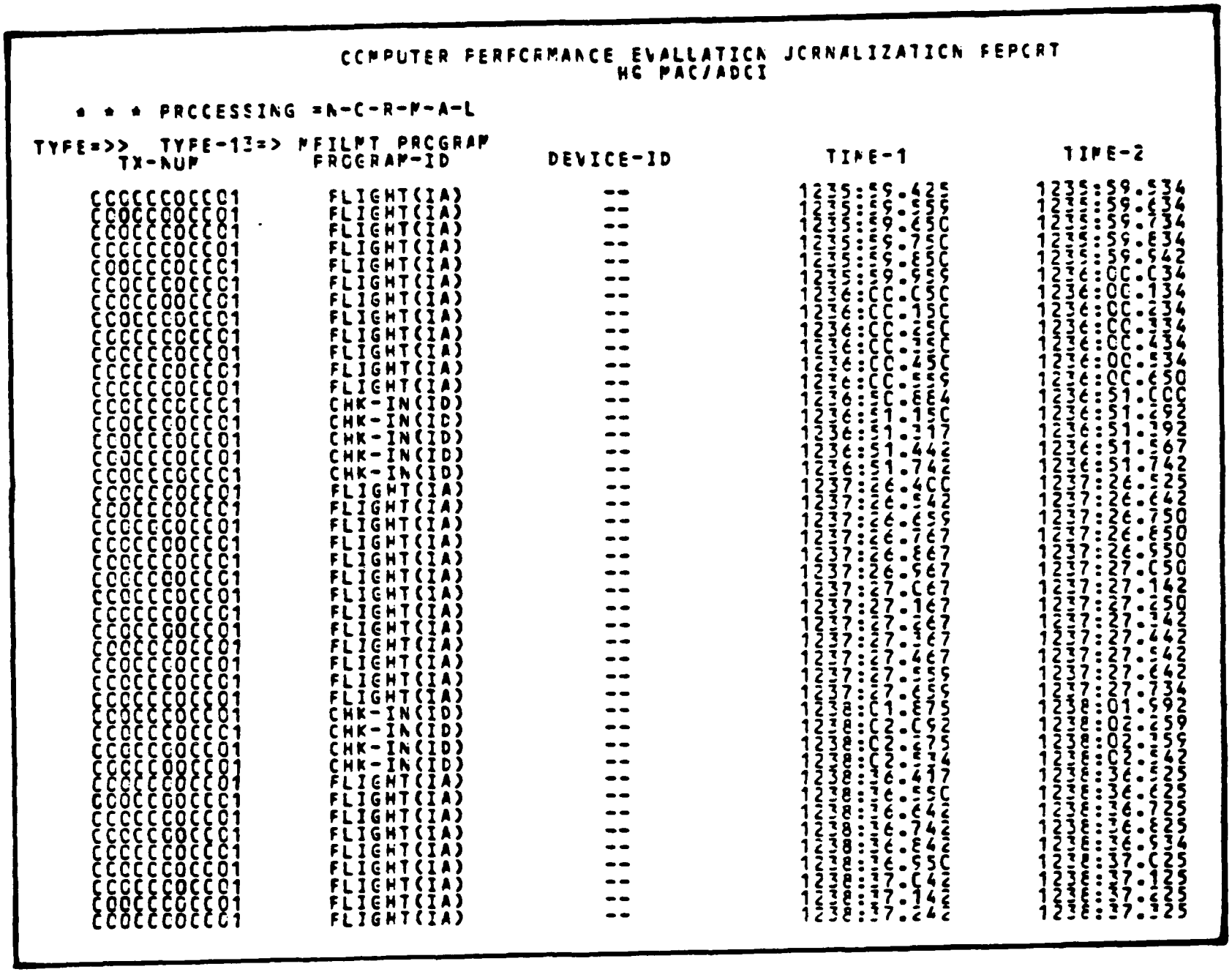
* * * PROCESSING =N-C-R-M-A-L

TYPE=>> TYPE-1C=> MIOH'S FOR INTERACTIVE DEVICES

TX-NUM	PROGRAM-ID	DEVICE-ID	TIME-1	TIME-2
*999999999	FLIGHT(IA)	LL	1235:42.575	1235:43.034
CCCCCCCCCC1	FLIGHT(IA)	LL	1235:58.7CC	1236:02.C75
CCCCCCCCCC1	*	LL	1236:18.7C5	1236:19.C42
CCCCCCCCCC1	CHK-IN(ID)	LL	1236:14.7C5	1236:15.C65
CCCCCCCCCC1	CHK-IN(ID)	LL	1236:51.7C5	1236:51.C69
CCCCCCCCCC1	FLIGHT(IA)	LL	1237:1C.7C5	1237:1C.C75
CCCCCCCCCC1	FLIGHT(IA)	LL	1237:25.7C5	1237:26.C75
CCCCCCCCCC1	CHK-IN(ID)	LL	1237:46.7C5	1237:46.C92
CCCCCCCCCC1	CHK-IN(ID)	LL	1238:C1.7C5	1238:04.C45
CCCCCCCCCC1	FLIGHT(IA)	LL	1238:19.7C5	1238:2C.1CC
CCCCCCCCCC1	FLIGHT(IA)	LL	1238:39.7C5	1238:41.125
CCCCCCCCCC1	FLIGHT(IA)	LL	1238:59.7C5	1239:C1.C42
CCCCCCCCCC1	FLIGHT(IA)	LL	1239:17.7C5	1239:2C.C42
CCCCCCCCCC1	FLIGHT(IA)	LL	1239:35.7C5	1239:44.C52
CCCCCCCCCC1	FLIGHT(IA)	LL	1239:54.7C5	1239:59.C42
CCCCCCCCCC1	FLIGHT(IA)	LL	1240:13.7C5	1240:21.C42
CCCCCCCCCC1	FLIGHT(IA)	LL	1240:33.7C5	1240:4C.C52
CCCCCCCCCC1	FLIGHT(IA)	LL	1240:52.7C5	1241:C1.C42
CCCCCCCCCC1	FLIGHT(IA)	LL	1241:12.7C5	1241:19.C42
CCCCCCCCCC1	FLIGHT(IA)	LL	1241:31.7C5	1241:37.C34
CCCCCCCCCC1	FLIGHT(IA)	LL	1241:51.7C5	1241:56.C39
CCCCCCCCCC1	FLIGHT(IA)	LL	1242:1C.7C5	1242:14.C59
CCCCCCCCCC1	MA-RES(IC)	LL	1242:29.7C5	1242:31.C89
CCCCCCCCCC1	MA-RES(IC)	LL	1242:49.7C5	1242:49.C89
CCCCCCCCCC1	CHK-IN(ID)	LL	1243:08.7C5	1243:06.159
CCCCCCCCCC1	CHK-IN(ID)	LL	1243:27.7C5	1243:24.159
CCCCCCCCCC1	CHK-IN(ID)	LL	1243:47.7C5	1243:42.15C
CCCCCCCCCC1	FLIGHT(IA)	LL	1244:C6.7C5	1243:58.1CC
CCCCCCCCCC1	FLIGHT(IA)	LL	1244:25.7C5	1244:17.C67
CCCCCCCCCC1	CHK-IN(ID)	LL	1244:45.7C5	1244:34.C67
CCCCCCCCCC1	CHK-IN(ID)	LL	1245:C4.7C5	1244:53.C25
CCCCCCCCCC1	CHK-IN(ID)	LL	1245:24.7C5	1245:13.15C
CCCCCCCCCC1	CHK-IN(ID)	LL	1245:43.7C5	1245:33.C67

		HIGH	LOW
***** >>>TOTALS:	NBR OF TRANSACTIONS CCC33	CCCC:19.45C (CCC25)	CCCC:0C.459 (CCCC1)

REC-TYPE AVERAGE
(CCCC:C2.05E)

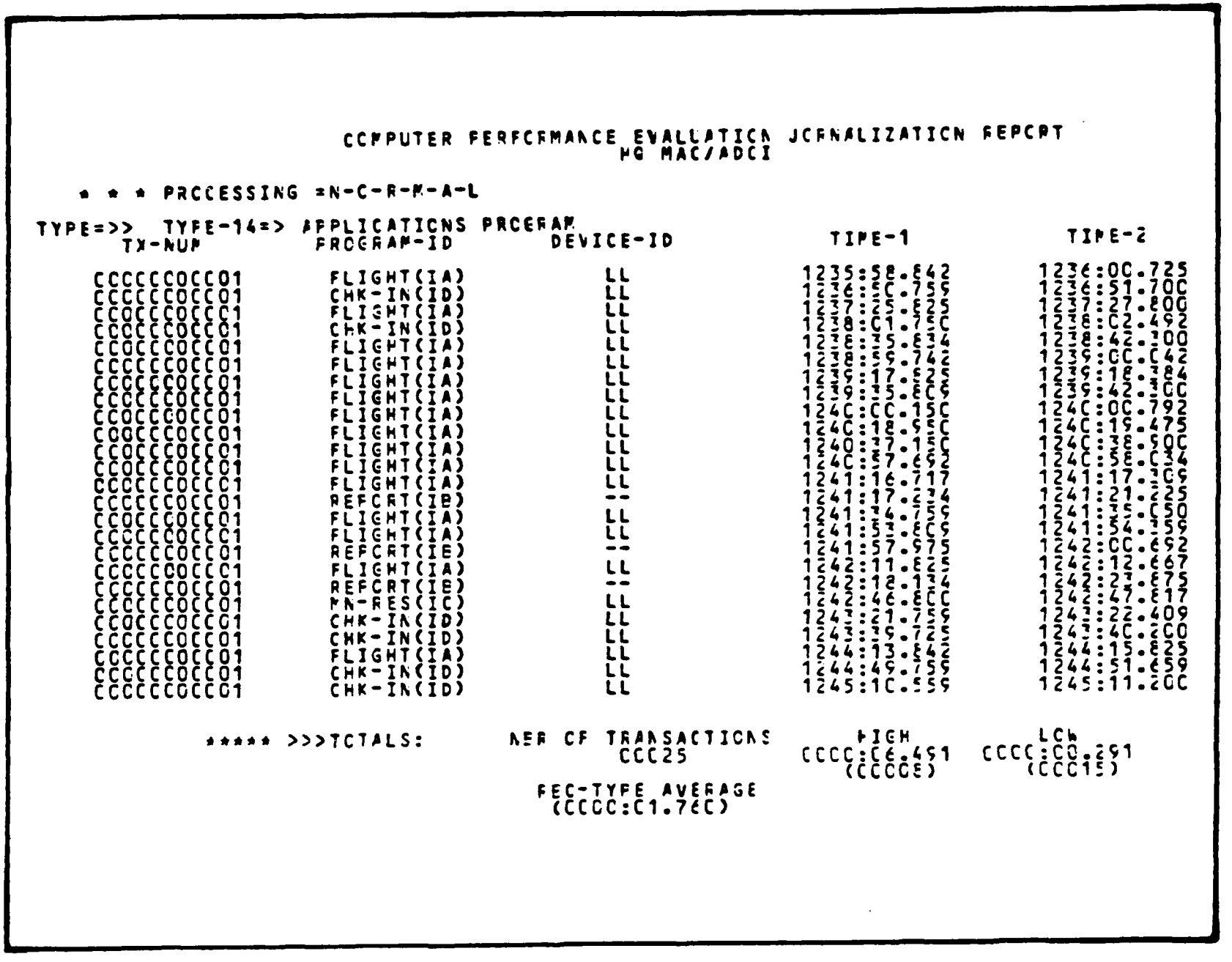


COMPUTER PERFORMANCE EVALUATION JORNALIZATION REPORT
HG MAC/ADCI

* * * PROCESSING =N-C-R-M-A-L

TYPE=>> TYPE-13=> MFILMT PROGRAM

TX-NUM	PROGRAM-ID	DEVICE-ID	TIME-1	TIME-2
0000000001	FLIGHT(IA)	--	1235:59.425	1235:59.534
0000000001	FLIGHT(IA)	--	1235:59.559	1235:59.634
0000000001	FLIGHT(IA)	--	1235:59.650	1235:59.734
0000000001	FLIGHT(IA)	--	1235:59.750	1235:59.834
0000000001	FLIGHT(IA)	--	1235:59.850	1235:59.942
0000000001	FLIGHT(IA)	--	1235:59.959	1236:00.034
0000000001	FLIGHT(IA)	--	1236:00.050	1236:00.134
0000000001	FLIGHT(IA)	--	1236:00.150	1236:00.234
0000000001	FLIGHT(IA)	--	1236:00.250	1236:00.334
0000000001	FLIGHT(IA)	--	1236:00.350	1236:00.434
0000000001	FLIGHT(IA)	--	1236:00.450	1236:00.534
0000000001	FLIGHT(IA)	--	1236:00.559	1236:00.650
0000000001	CHK-IN(ID)	--	1236:50.884	1236:51.000
0000000001	CHK-IN(ID)	--	1236:51.150	1236:51.292
0000000001	CHK-IN(ID)	--	1236:51.317	1236:51.392
0000000001	CHK-IN(ID)	--	1236:51.442	1236:51.567
0000000001	CHK-IN(ID)	--	1236:51.742	1236:51.742
0000000001	FLIGHT(IA)	--	1237:26.400	1237:26.525
0000000001	FLIGHT(IA)	--	1237:26.542	1237:26.642
0000000001	FLIGHT(IA)	--	1237:26.659	1237:26.750
0000000001	FLIGHT(IA)	--	1237:26.767	1237:26.850
0000000001	FLIGHT(IA)	--	1237:26.867	1237:26.950
0000000001	FLIGHT(IA)	--	1237:26.967	1237:27.050
0000000001	FLIGHT(IA)	--	1237:27.067	1237:27.142
0000000001	FLIGHT(IA)	--	1237:27.167	1237:27.250
0000000001	FLIGHT(IA)	--	1237:27.267	1237:27.342
0000000001	FLIGHT(IA)	--	1237:27.367	1237:27.442
0000000001	FLIGHT(IA)	--	1237:27.467	1237:27.542
0000000001	FLIGHT(IA)	--	1237:27.559	1237:27.642
0000000001	FLIGHT(IA)	--	1237:27.659	1237:27.734
0000000001	CHK-IN(ID)	--	1238:01.875	1238:01.992
0000000001	CHK-IN(ID)	--	1238:02.092	1238:02.259
0000000001	CHK-IN(ID)	--	1238:02.275	1238:02.359
0000000001	CHK-IN(ID)	--	1238:02.534	1238:02.542
0000000001	FLIGHT(IA)	--	1238:36.417	1238:36.525
0000000001	FLIGHT(IA)	--	1238:36.550	1238:36.625
0000000001	FLIGHT(IA)	--	1238:36.642	1238:36.725
0000000001	FLIGHT(IA)	--	1238:36.742	1238:36.825
0000000001	FLIGHT(IA)	--	1238:36.842	1238:36.934
0000000001	FLIGHT(IA)	--	1238:36.950	1238:37.025
0000000001	FLIGHT(IA)	--	1238:37.042	1238:37.125
0000000001	FLIGHT(IA)	--	1238:37.142	1238:37.225
0000000001	FLIGHT(IA)	--	1238:37.242	1238:37.325



COMPUTER PERFORMANCE EVALUATION JORNALIZATION REPORT
HQ MAC/ADCI

* * * PROCESSING =N-O-R-M-A-L

TYPE=>> TYPE-14=> APPLICATIONS PROGRAM

TX-NUM	PROGRAM-ID	DEVICE-ID	TIME-1	TIME-2
0000000001	FLIGHT(IA)	LL	1235:58.842	1236:00.725
0000000001	CHK-IN(ID)	LL	1236:50.759	1236:51.700
0000000001	FLIGHT(IA)	LL	1237:25.825	1237:27.800
0000000001	CHK-IN(ID)	LL	1238:01.750	1238:02.492
0000000001	FLIGHT(IA)	LL	1238:35.834	1238:42.300
0000000001	FLIGHT(IA)	LL	1238:59.742	1239:00.042
0000000001	FLIGHT(IA)	LL	1239:17.825	1239:18.384
0000000001	FLIGHT(IA)	LL	1239:35.809	1239:42.300
0000000001	FLIGHT(IA)	LL	1240:00.150	1240:00.792
0000000001	FLIGHT(IA)	LL	1240:18.950	1240:19.475
0000000001	FLIGHT(IA)	LL	1240:37.150	1240:38.900
0000000001	FLIGHT(IA)	LL	1240:57.692	1240:58.034
0000000001	FLIGHT(IA)	LL	1241:16.717	1241:17.309
0000000001	REPORT(IB)	--	1241:17.234	1241:21.225
0000000001	FLIGHT(IA)	LL	1241:34.759	1241:35.050
0000000001	FLIGHT(IA)	LL	1241:53.809	1241:54.359
0000000001	REPORT(IB)	--	1241:57.975	1242:00.692
0000000001	FLIGHT(IA)	LL	1242:11.825	1242:12.667
0000000001	REPORT(IB)	--	1242:18.134	1242:23.875
0000000001	MN-RES(IC)	LL	1242:46.800	1242:47.817
0000000001	CHK-IN(ID)	LL	1243:21.759	1243:22.409
0000000001	CHK-IN(ID)	LL	1243:39.725	1243:40.200
0000000001	FLIGHT(IA)	LL	1244:13.842	1244:15.825
0000000001	CHK-IN(ID)	LL	1244:49.759	1244:51.659
0000000001	CHK-IN(ID)	LL	1245:10.559	1245:11.200

***** >>>TOTALS: NER OF TRANSACTIONS 00025

HIGH 0000:06.491 (00008)

LOW 0000:00.291 (00015)

REC-TYPE AVERAGE (0000:01.760)

Bibliography

1. Brannigan, Vincent. "Acceptance Testing - The Critical Problem in Software Acquisition," IEEE Transactions on Biomedical Engineering, 32: 295-299 (April 1985).

2. Citron, Andrew. "A Software Review Method That Really Works," BYTE 437-440 (January 1984).

3. DeJesus, Enrique G. and Craig J. Riesberg. "Automating Configuration Management," Proceedings of the Annual Computer Related Information Systems Symposium. 3.1-3.22. Department of Computer Science, United States Air Force Academy, 1983.

4. DeJesus, Enrique G. and Craig J. Riesberg. "Automating Configuration Management," Proceedings of the Computer Performance Evaluation Users Group 19th Meeting. 187-196. United States Governament Printing Office, Washington, D.C., 1983.

5. DeJesus, Enrique G. and Craig J. Riesberg. "Automating Configuration Management," Proceedings of the 1983 Computer Measurement Group International Conference. 222-230. Computer Measurement Group, Phoenix, Arizona, 1983.

6. Department of the Air Force. Remote Terminal Emulator Site Configuration Manual. Joint Data Systems Support Center, Reston, Va., 13 January 1984.

7. Digilog, Inc. Operator's Instruction Manual for DLM IV Data Line Monitor. Montgomeryville, Pa., March 1983.

8. Evans, Ralph R. "Reliability Testing," IEEE Transactions on Reliability, 32: 337 (October 1984).

9. Fox, Joseph M. Software and its Development. Englewood Cliffs: Prentice-Hall, Inc., 1982.

10. Honeywell Information Systems, Inc. GCOS 6 MOD 400 System Concepts. Westwood, Mass., June 1984.

11. Honeywell Information Systems, Inc. GCOS 6 MOD 400 System Messages. Westwood, Mass., December 1982.

12. Honeywell Information Systems, Inc. System Building and Administration. Westwood, Mass., October 1984.

13. Kubat, Peter and Harvey S. Koch. "Pragmatic Testing Protocols to Measure Software Reliability, IEEE Transactions on Reliability, 32: 338-341 (October 1984).

14. Morrocco, John. "Coming Up Short in Software," AIR FORCE Magazine, 64-69 (February 1987).

15. Plessey MicroSystems. Inquistor. Product Brochure for External Test Driver/Remote Terminal Emulator. Emulation Engineering Group, Plessey MicroSystems, Rockville, Md., undated.

16. Pressman, Roger S. Software Engineering: A Practitioner's Approach. New York: McGraw-Hill Book Company, 1982.

17. Zelkowitz, Marvin V., Alan C. Shaw, and John D. Gannon. Principles of Software Engineering and Design. Englewood Cliffs: Prentice-Hall, Inc., 1979.

VITA

Captain Craig J. Riesberg was born 9 November 1949 in Carroll, Iowa. He graduated from Kuemper High School in 1967. He enlisted in the United States Air Force 17 August, 1971. He attended the University of Texas in San Antonio and graduated in 1981, receiving the degree of Bachelor of Science in Mathematics and Computer Science. He was commissioned a Second Lieutenant in May 1981.

Captain Riesberg was assigned to the 1500 Computer Services Squadron, as a Software Validation and Computer Performance Evaluation Officer prior to his assignment at the School of Engineering, Air Force Institute of Technology in May of 1986.

Permanent address: 435 Ridgewood Drive
Carroll, Iowa, 51401

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS AD-A188 832
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/MA/87D-5	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Hq MAC, ACD 8b OFFICE SYMBOL (If applicable) ACD/AUC	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Scott AFB, ILL. 62225	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
MODIFICATION, IMPLEMENTATION, AND EVALUATION OF A REMOTE TERMINAL EMULATOR AS A SOFTWARE VALIDATION AND STRESS TESTING TOOL

12. PERSONAL AUTHOR(S)
Craig J. Riesberg, B.S., Captain, US Air Force

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
MS Thesis	FROM _____ TO _____	1987, December	108

16. SUPPLEMENTARY NOTATION

Lynn W. Dana 31 Dec 87
Dean for Research and Professional Development
Air Force Institute of Technology
Wright-Patterson AFB OH 45433

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Remote Terminal Emulator, Software Validation, Stress Testing, Software Testing

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This project involved the modification, implementation and evaluation of a remote terminal emulator (RTE) as a software validation and stress testing tool. The work, sponsored by the Directorate of Integration and Standardization, Military Airlift Command, utilized a Honeywell DPS-6 Model 95 minicomputer as the host for the RTE application package.

Considerable background information is provided about the sponsor's release environment to show the sponsor's need for a reliable testing tool. Information is also presented about acceptance and validation testing.

The emphasis of this research centered on a RTE made available by the Joint Data System Support Center. This software package, capable of emulating multiple users, needed major modifications before implementation on the sponsor's equipment. After successful implementation, the study examined two areas of use for the sponsor.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Panna B. Nagarsenker	22b TELEPHONE (Include Area Code) 513-255-7210 / 22c OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 Previous editions are obsolete

19. The modified package was found to be a very good tool for software validation. Comparisons are made between the RTE and the sponsor's internal test driver in the areas of scenario collection and management. The two tools are also compared during the emulation phase of software validation.

The RTE package was also examined as a stress testing tool. Several problems in the RTE application software which inhibit its use as a stress testing tool are discussed.

END

FILMED

MARCH, 1988

DTIC